STD-LIB
Library Reference

Section 5

BASIC LANGUAGE LIBRARY

CONTENTS **PAGE**

Introduction **5-1**

Analog Devices RTI-1225 Analog Input/Output Card **5-1225-1**

AD1225AIN 5-1225-4
AD1225AOUT 5-1225-5
AD1225INIT 5-1225-7
AD1225RESET 5-1225-11
AD1225TEST 5-1225-12

Analog Devices RTI-1226 Analog Input Card **5-1226-1**

AD1226AIN 5-1226-4
AD1226INIT 5-1226-5
AD1226RESET 5-1226-8
AD1226TEST 5-1226-9

Analog Devices RTI-1260 Analog Input Card **5-1260-1**

AD1260AIN 5-1260-4
AD1260INIT 5-1260-5
AD1260RESET 5-1260-8
AD1260TEST 5-1260-9

Analog Devices RTI-1262 Analog Output Card **5-1262-1**

AD1262AMPS 5-1262-4
AD1262AOUT 5-1262-6
AD1262INIT 5-1262-8
AD1262RESET 5-1262-11

Pro-Log 75XX, 76XX, and 79XX Input/Output Cards **5-7000-1**

PL7000INIT 5-7000-3
PL7000RDBIT 5-7000-5
PL7000RDPORT 5-7000-6
PL7000RESETPORT 5-7000-7
PL7000WRTBIT 5-7000-8
PL7000WRTBITRD 5-7000-10
PL7000WRTPORT 5-7000-12
PL7000WRTPORTRD 5-7000-14
7501 DC Driver Card 5-7000-16
7502 SPST Relay Output Card 5-7000-16
7503 Optoisolated Input Card 5-7000-17
7504 Triac Output Card 5-7000-17
7506 Optoisolated Input Card 5-7000-18
7601A TTL Input/Output Port Card 5-7000-19
7602A TL Output Port Card 5-7000-20
7603A TTL Input Port Card 5-7000-20
7604A TTL Input/Output Card 5-7000-21
7605 Programmable TTL I/O Card 5-7000-22
7614 65-Line TTL I/O Card 5-7000-23
7904 Decoded I/O Utility Card 5-7000-24
7914 Decoded I/O Utility Card 5-7000-25

Pro-Log 7303 Keyboard Display Card **5-7303-1**

PL7303CHRSHFT 5-7303-3
PL7303CHRWR 5-7303-4
PL7303CURSOR 5-7303-5
PL7303GETKEY 5-7303-6
PL7303GETROCKR 5-7303-7
PL7303INIT 5-7303-8
PL7303KEYFETCH 5-7303-10
PL7303KEYSTAT 5-7303-11

PL7303MESSAGE 5-7303-12
PL7303RCKRSTAT 5-7303-13
PL7303RESET 5-7303-14
PL7303SETLEDS 5-7303-15
PL7303TEST 5-7303-16
PL7303WAIT 5-7303-17
PL7303WORD 5-7303-18

Pro-Log 7312/7314A Communications Card **5-7314-1**

PL7314BUFFLUSH 5-7314-3
PL7314CHANRST 5-7314-4
PL7314INIT 5-7314-6
PL7314INTEN 5-7314-8
PL7314RCV 5-7314-10
PL7314RCVCHK 5-7314-11
PL7314SETBUF 5-7314-12
PL7314TEST 5-7314-14
PL7314XMT 5-7314-15
PL7314XMTBRK 5-7314-16
PL7314XMTCHK 5-7314-17
PL7314XMTIMD 5-7314-18

Pro-Log 7507/7508 TTL Input/Output Card **5-7508-1**

PL7508INIT 5-7508-2
PL7508RACKPWR 5-7508-4
PL7508READMOD 5-7508-5
PL7508RESET 5-7508-6
PL7508SETMOD 5-7508-7
PL7508TEST 5-7508-8

Pro-Log 7700 Memory Diagnostic Routines **5-7700-1**

PL7700CHECKSUM 5-7700-2
PL7700MEMTEST 5-7700-3

Pro-Log 7863 CPU Card **5-7863-1**

PL7863CHKMARK 5-7863-3
PL7863GETDATE 5-7863-4
PL7863GETTIME 5-7863-5
PL7863GETWATCH 5-7863-6
PL7863INIT 5-7863-7
PL7863SETALARM 5-7863-9
PL7863SETMARK 5-7863-11
PL7863TEST 5-7863-13
PL7863WAIT 5-7863-14
PL7863WEEKDAY 5-7863-15

Pro-Log 7888 CPU Card **5-7888-1**

PL7888CHKMARK 5-7888-3
PL7888GETDATE 5-7888-4
PL7888GETTIME 5-7888-5
PL7888GETWATCH 5-7888-6
PL7888INIT 5-7888-7
PL7888SETALARM 5-7888-9
PL7888SETMARK 5-7888-11
PL7888TEST 5-7888-13
PL7888WAIT 5-7888-14
PL7888WEEKDAY 5-7888-15

Section 5

BASIC LANGUAGE LIBRARY

INTRODUCTION

This section contains the STD-LIB routines for application programs written in Microsoft BASIC.

Your BASIC application program should be linked with the STD-LIB BASIC language version, **STDLIBB.LIB**. Refer to Section 2, Operation and Programming, for a description of how to link the BASIC version of STD-LIB with your application program.

Analog Devices
RTI-1225
Analog Input/Output Card
BASIC library

The Analog Devices RTI-1225 Analog Input/Output Card is a general-purpose, low-cost analog I/O card. The RTI-1225 card uses an analog multiplexer to select between analog input channels that are connected to a 10-bit A/D converter. The input channels can be configured to be 16 single-ended or 8 differential inputs. There are two 8-bit D/A converters providing two output channels.

The STD-LIB RTI-1225 software modules are as follows:

AD1225AIN:	Reads an input voltage and converts the A/D converter binary value to a reading rounded to the nearest millivolt (based on the default hardware configuration or the configuration set by AD1225INIT).
AD1225AOUT:	Transforms the output voltage requested in the call from millivolts to the nearest D/A code and writes an output voltage.
AD1225INIT:	Initializes the library software with the user-selected hardware configuration of the card. If this routine has been invoked properly, a call is made to the AD1225RESET routine.
AD1225RESET:	Performs a software reset of the card. The multiplexer is set to select input channel 0 and wait for any pending conversions to complete or a conversion busy bit timeout. The D/A input registers are set to output a voltage of 0V.

AD1225TEST:	Performs a self-test on one card. The routine reads a voltage from the card and passes self-test if there is no timeout error.

NOTE: The RTI-1225 as-shipped mapping is at hexadecimal base address FFF0 in the memory map. To be compatible with the library, the card needs to be port mapped with an 8-bit address. The default 8-bit base address for the library is hexadecimal B0 which results in the card using ports BB through BF.

The Analog Devices convention is to define the base address of a card as an address with the last four bits being zero. The library has followed this convention. This means that an RTI-1225 card that has been remapped to the library default would be defined as a base address of 1B0 in the initialize command, but the physical ports of the card are actually located at 1BB through 1BF. When referring to contiguous mapping for RTI-1225 cards, the cards are mapped as though they occupied 16 ports per card, even though they only physically occupy five ports per card.

To change an RTI-1225 card from its as-shipped mapping to the library default mapping, make the following changes to the jumpers:

remove 59-60, 62-63, 51-52
install 61-63, 58-59, 53-55, 15-16

For technical questions concerning STD-LIB software for the Analog Devices RTI-1225, contact Pro-Log customer service. For technical questions concerning the operation of the Analog Devices RTI-1225 hardware, contact the appropriate Analog Devices customer service as follows:

STATE	PHONE NUMBER
Alaska Arizona California (southern) Colorado	(714)-641-9391 Direct Sales Office
California (northern) Utah	(408)-947-0633 Direct Sales Office
Delaware New Jersey Pennsylvania	(215)-643-7790 Direct Sales Office
Maine Massachusetts New Hampshire Rhode Island	(617)-329-4700 Factory
Illinois Indiana Iowa Kansas Kentucky Missouri North Dakota South Dakota Wisconsin	(312)-653-5000 Direct Sales Office
All Other States	(617)-329-4700 Factory

Structure: **CALL AD1225AIN(VOLTAGE%, CARDNO%,CHANNEL%)**

Description: This routine reads an input voltage from the selected channel. The binary reading of the A/D converter is translated to a binary reading in millivolts based on the user-selected input configuration.

Input Parameters: **CARDNO%** is an integer value specifying the card from which the reading is to be taken (0 - 21).

CHANNEL% is an integer value specifying the input channel to be read (0 - 15).

Returned Value: **VOLTAGE%** contains the analog voltage read in millivolts (-10000 to + 9990) or contains an error code:

-32768 = channel number out of range
-32767 = board number out of range
-32766 = timeout error during conversion

Example: The following BASIC program segment could be written to read an analog voltage from input channel 2 of the fourth card in the system.

```
40  CARDNO%=3
50  CHANNEL%=2
60  CALL AD1225AIN(VOLTAGE%,
CARDNO%,CHANNEL%)
```

AD1225AOUT

Structure:

CALL AD1225AOUT(ARROR%, CARDNO%,CHANNEL%,VOLTAGE%)

Description:

This routine outputs a voltage to the selected channel. The selected voltage is input in millivolts and converted to a binary program value for the D/A converter based on the user-selected output configuration for the channel.

Input Parameters:

CARDNO% is an integer value specifying the card from which the voltage is to be output (0 - 21).

CHANNEL% is an integer value specifying the channel from which the voltage is to be output (0 or 1).

VOLTAGE% contains the analog voltage to be output in millivolts (-10000 to + 9961).

Returned Value:

ARROR% contains the returned integer error code:

0 = no error
-1 = channel number out of range
-2 = board number out of range
-3 = voltage parameter too large (clipped to maximum value)
-4 = voltage parameter too small (clipped to minimum value)

Example:

The following BASIC program segment could be written to output an analog voltage of 2V from output channel 1 of the fourth card in the system.

```
40 CARDNO%=3
50 CHANNEL%=1
```

```
60 VOLTAGE%=2000
70 CALL AD1225AOUT(ARROR%,
CARDNO%,CHANNEL%,VOLTAGE%)
```

Structure:

CALL AD1225INIT(ARROR%, BASENO%,BASEADR%,NOCARDS%, INCHAN%,ADCONF%,DA0CONF%, DA1CONF%)

Description:

This routine initializes the RTI-1225 library routines with the user-selected hardware configuration of the card if different from the default condition. The card default initialization is for one RTI-1225 card in a system with a base address of 1B0, configured for 16 single-ended inputs of ±10V two's complement and both outputs ±10V and two's complement. There can be up to 22 of these cards in a system. When using more than one card, all cards with the same configuration are mapped contiguously and are initialized with the same routine call. If there are no parameter errors, the card is software reset with a call to the AD1225RESET routine.

Input Parameters:

BASENO% is an integer value specifying the base number of the group of contiguous cards configured by this command (0 - 21).

BASEADR% is a hexadecimal integer value specifying the base address of the card. This is the lowest port address of the contiguously mapped RTI-1225 cards. The library routines use a 10-bit address. The RTI-1225 card should be mapped for an 8-bit I/O address. In software, set the two most significant bits of the 10-bit address to 01 (for example, the RTI-1225 address 20 is entered as 120). The least significant four bits of the address must be 0. The routines do not work if the card is memory mapped.

NOCARDS% is an integer value specifying the number of contiguously mapped cards specified by this command (1 - 22).

INCHAN% is the input multiplexer configuration:

01 = single-ended
02 = differential
03 = pseudo-differential

ADCONF% is the A/D input range and output code:

00 = 0V to +10V, straight binary
01 = -5V to +5V, two's complement
02 = -5V to +5V, offset binary
03 = -10V to +10V, two's complement
04 = -10V to +10V, offset binary

DA0CONF% is the D/A channel 0 output range and input code:

00 = 0V to +10V, straight binary
01 = -5V to +5V, two's complement
02 = -5V to +5V, offset binary
03 = -10V to +10V, two's complement
04 = -10V to +10V, offset binary

DA1CONF% is the D/A channel 1 output range and input code:

00 = 0V to +10V, straight binary
01 = -5V to +5V, two's complement
02 = -5V to +5V, offset binary
03 = -10V to +10V, two's complement
04 = -10V to +10V, offset binary

Returned Value:

ARROR% contains the returned integer error code:

0 = no error
-1 = board number out of range
-2 = invalid card count
  (NOCARDS% >22, or the

number of cards causes the base address of the last card to be less than that of the first card (wraparound)).
-3 = invalid multiplexer configuration
-4 = invalid A/D input range/output code
-5 = invalid D/A channel 0 0 input code/output range
-6 = invalid D/A channel 1 input code/output range
-7 = timeout error during reset

Example:

The following BASIC program segment could be written to initialize the library for a group of four RTI-1225 cards after a group of three differently configured RTI-1225 cards have already been initialized. This group of cards is mapped contiguously beginning at 8-bit I/O address &H20. The cards are configured for differential inputs, 0 to +10V input range with straight binary code, and both output channels for 0 to +10V output range with straight binary code.

```
40  BASENO%=3
50  BASEADR%=&H120
60  NOCARDS%=4
70  INCHAN%=02
80  ADCONF%=00
90  DA0CONF%=00
100 DA1CONF%=00
110 CALL AD1225INIT(ARROR%,
BASENO%,BASEADR%,NOCARDS%,
INCHAN%,ADCONF%,DA0CONF%,
DA1CONF%)
```

AD1225RESET

Structure:	**CALL AD1225RESET(ARROR%)**
Description:	This routine software resets all the RTI-1225 cards in the system. The multiplexer is set to select input channel 0, a wait for any pending conversions to complete or a timeout is performed. The D/A input registers are set to output a voltage of 0V. Note that the D/A output voltage may differ between hardware and software resets, depending on the user-selected hardware configuration.
Input Parameters:	None
Returned Value:	ARROR% contains the returned integer error code: 0 = no error -7 = timeout error during reset
Example:	The following BASIC program segment resets all RTI-1225 cards in a system. `40 CALL AD1225RESET(ARROR%)`

Structure:	CALL AD1225TEST(ARROR%, CARDNO%)
Description:	This routine self-tests an RTI-1225 card. A voltage is read from the A/D converter and self-test is passed if there is no timeout error.
Input Parameters:	CARDNO% is an integer value specifying the card to be tested (0 - 21).
Returned Value:	ARROR% contains the returned integer error code: 0 = no error -1 = invalid card selected -7 = timeout error
Example:	The following BASIC program segment tests the second RTI-1225 card in a system.

```
40 CARDNO%=1
50 CALL AD1225RESET(ARROR%,
CARDNO%)
```

Analog Devices
RTI-1226
Analog Input Card
BASIC Library

The Analog Devices RTI-1226 Analog Input Card is a general-purpose, low-cost analog I/O card. The RTI-1226 card uses an analog multiplexer to select among analog input channels that are connected to a 10-bit A/D converter. The input channels can be configured to be 16 single-ended or 8 differential inputs.

The STD-LIB RTI-1226 software modules are as follows:

AD1226AIN:	Reads an input voltage and converts the A/D converter binary value to a reading rounded to the nearest millivolt (based on the default hardware configuration or the configuration set by AD1226INIT).
AD1226INIT:	Initializes the library software with the user-selected hardware configuration of the card. If this routine has been invoked properly, a call is made to the AD1226RESET routine.
AD1226RESET:	Performs a software reset of the card. The multiplexer is set to select input channel 0 and wait for any pending conversions to complete or a conversion busy bit timeout.
AD1226TEST:	Performs a self-test on one card. The routine reads a voltage from the card and passes self-test if there is no timeout error.

NOTE: The RTI-1226 as-shipped mapping is at hexadecimal base address FFF0 in the memory map. To be compatible with the library, the card needs to be port mapped with an 8-bit address. The default 8-bit base address for the library is hexadecimal B0 which results in the card using ports BB through BD.

The Analog Devices convention is to define the base address of a card as an address with the last four bits being zero. The library has followed this convention. This means that an RTI-1226 card that has been remapped to the library default would be defined as a base address of 1B0 in the initialize command, but the physical ports of the card are actually located at 1BB through 1BD. When referring to contiguous mapping for RTI-1226 cards, the cards are mapped as though they occupied 16 ports per card, even though they only physically occupy three ports per card.

To change an RTI-1226 card from its as-shipped mapping to the library default mapping, make the following changes to the jumpers:

remove 59-60, 62-63, 51-52
install 61-63, 58-59, 53-55, 15-16

For technical questions concerning STD-LIB software for the Analog Devices RTI-1226, contact Pro-Log customer service. For technical questions concerning the operation of the Analog Devices RTI-1226 hardware, contact the appropriate Analog Devices customer service as follows:

STATE	PHONE NUMBER
Alaska	
Arizona	(714)-641-9391
California (southern)	Direct Sales Office
Colorado	
California (northern)	(408)-947-0633
Utah	Direct Sales Office
Delaware	(215)-643-7790
New Jersey	Direct Sales Office
Pennsylvania	
Maine	
Massachusetts	(617)-329-4700
New Hampshire	Factory
Rhode Island	
Illinois	
Indiana	
Iowa	
Kansas	
Kentucky	(312)-653-5000
Missouri	Direct Sales Office
North Dakota	
South Dakota	
Wisconsin	
All Other States	(617)-329-4700
	Factory

Structure:	**CALL AD1226AIN(VOLTAGE%,CARDNO%, CHANNEL%)**
Description:	This routine reads an input voltage from the selected channel. The binary reading of the A/D converter is translated to a binary reading in millivolts based on the user-selected input configuration.
Input Parameters:	**CARDNO%** is an integer value specifying the card from which the reading is to be taken (0 - 21). **CHANNEL%** is an integer value specifying the input channel to be read (0 - 15).
Returned Value:	**VOLTAGE%** contains the analog voltage read in millivolts (-10000 to + 9990) or contains an error code: -32768 = channel number out of range -32767 = board number out of range -32766 = timeout error during conversion
Example:	The following BASIC program segment could be written to read an analog voltage from input channel 2 of the fourth card in the system.

```
40 CARDNO%=3
50 CHANNEL%=2
60 CALL AD1226AIN(VOLTAGE%,
CARDNO%, CHANNEL%)
```

Structure: **CALL AD1226INIT(ARROR%,BASENO%, BASEADR%,NOCARDS%,INCHAN%, ADCONF%)**

Description: This routine initializes the RTI-1226 library routines with the user-selected hardware configuration of the card if different from the default condition. The card default initialization is for one RTI-1226 card in a system with a base address of 1B0, configured for 16 single-ended inputs of ±10V two's complement and both outputs ±10V and two's complement. There may be up to 22 of these cards in a system. When using more than one card, all cards with the same configuration are mapped contiguously and are initialized with the same routine call. If there are no parameter errors, the card is software reset with a call to the AD1226RESET routine.

Input Parameters: **BASENO%** is an integer value specifying the base number of the group of contiguous cards configured by this command (0 - 21).

BASEADR% is a hexadecimal integer value specifying the base address of the card. This is the lowest port address of the contiguously mapped RTI-1226 cards. The library routines use a 10-bit address. The RTI-1226 card should be mapped for an 8-bit I/O address. In software, set the most significant bits of the 10-bit address to 01 (for example, the RTI-1226 address 20 is entered as 120). The least significant four bits of the address must be 0. The routines do not work if the card is memory mapped.

NOCARDS% is an integer value specifying the number of contiguously mapped cards specified by this command (1 - 22).

INCHAN% is the input multiplexer configuration:

01 = single-ended
02 = differential
03 = pseudo-differential

ADCONF% is the A/D input range and output code:

00 = 0V to +10V, straight binary
01 = -5V to +5V, two's complement
02 = -5V to +5V, offset binary
03 = -10V to +10V, two's complement
05 = -10V to +10V, offset binary

Returned Value:

ARROR% contains the returned integer error code:

0 = no error
-1 = board number out of range
-2 = invalid card count
    (NOCARDS% >22 or the number of cards causes the base address of the last card to be less than that of the first card (wraparound))
-3 = invalid multiplexer configuration
-4 = invalid A/D input range/output code
-7 = timeout error during reset

Example:

The following BASIC program segment could be written to initialize the library for a group of four RTI-1226 cards after a group of three differently configured RTI-1226 cards have already been initialized. This group of cards is mapped contiguously beginning at 8-bit I/O address 20. The cards are configured for differential inputs, 0 to +10V input range with straight binary code.

```
40 BASENO%=3
50 BASEADR%=&H120
60 NOCARDS%=4
70 INCHAN%=2
80 ADCONF%=00
90 CALL AD1226INIT(ARROR%,
BASENO%, BASEADR%,NOCARDS%,
INCHAN%, ADCONF%)
```

Structure:	**CALL AD1226RESET(ARROR%)**
Description:	This routine software resets all the RTI-1226 cards in the system. The multiplexer is set to select input channel 0, a wait for any pending conversions to complete or a timeout is performed.
Input Parameters:	None
Returned Value:	**ARROR%** contains the returned integer error code: 0 = no error -7 = timeout error during reset
Example:	The following BASIC program segment resets all RTI-1226 cards in a system. `CALL AD1226RESET(ARROR%)`

Structure: CALL AD1226TEST(ARROR%,CARDNO%)

Description: This routine self-tests an RTI-1226 card. A voltage is read from the A/D converter and self-test is passed if there is no timeout error.

Input Parameters: CARDNO% is an integer value specifying the card to be tested (0 - 21).

Returned Value: ARROR% contains the returned integer error code:

 0 = no error
-1 = invalid card selected
-7 = timeout error

Example: The following BASIC program segment tests the second RTI-1226 card in a system.

```
40 CARDNO%=1
50 CALL AD1226TEST(ARROR%,
CARDNO%)
```

Analog Devices
RTI-1260
Analog Input Card
BASIC library

The Analog Devices RTI-1260 Analog Input Card is a general-purpose, low-cost analog I/O card. The RTI-1260 card uses an analog multiplexer to select among analog input channels that are connected to a 12-bit A/D converter. The input channels can be configured to be 32 single-ended or 16 differential inputs.

The STD-LIB RTI-1260 software modules are as follows:

AD1260AIN:	Reads an input voltage and converts the A/D converter binary value to a reading rounded to the nearest microvolt (based on the default hardware configuration or the configuration set by AD1260INIT).
AD1260INIT:	Initializes the library software with the user-selected hardware configuration of the card.
AD1260RESET:	Performs a software reset of the card. The multiplexer is set to select input channel 0 and wait for any pending conversions to complete or a conversion busy bit timeout.
AD1260TEST:	Performs a self-test on one card. The routine reads a voltage from the card and passes self-test if there is no timeout error.

NOTE: The RTI-1260 as-shipped mapping is at hexadecimal base address FFF0 in the memory map. To be compatible with the library, the card needs to be port mapped with an 8-bit address. The default 8-bit base address for the library is hexadecimal 30 which results in the card using ports 3B through 3D.

The Analog Devices convention is to define the base address of a card as an address with the last four bits being zero. The library has followed this convention. This means that an RTI-1260 card that has been remapped to the library default would be defined as a base address of 130 in the initialize command, but the physical ports of the card are actually located at 13B through 13D. When referring to contiguous mapping for RTI-1260 cards, the cards are mapped as though they occupied 16 ports per card, even though they only physically occupy three ports per card.

To change an RTI-1260 card from its as-shipped mapping to the library default mapping, make the following changes to the jumpers:

remove 48-50, 51-52, 44-45
install 49-50, 52-53, 45-47, 15-16, 7-8

For technical questions concerning STD-LIB software for the Analog Devices RTI-1260, contact Pro-Log customer service. For technical questions concerning the operation of the Analog Devices RTI-1260 hardware, contact the appropriate Analog Devices customer service as follows:

STATE	PHONE NUMBER
Alaska Arizona California (southern) Colorado	(714)-641-9391 Direct Sales Office
California (northern) Utah	(408)-947-0633 Direct Sales Office
Delaware New Jersey Pennsylvania	(215)-643-7790 Direct Sales Office
Maine Massachusetts New Hampshire Rhode Island	(617)-329-4700 Factory
Illinois Indiana Iowa Kansas Kentucky Missouri North Dakota South Dakota Wisconsin	(312)-653-5000 Direct Sales Office
All Other States	(617)-329-4700 Factory

AD1260AIN

Structure:

CALL **AD1260AIN**(VOLTAGE,CARDNO%, CHANNEL%)

Description:

This routine reads an input voltage from the selected channel. The binary reading of the A/D converter is translated to a binary reading in microvolts based on the user-selected input configuration.

Input Parameters:

CARDNO% is an integer value specifying the card from which the reading is to be taken (0 - 21).

CHANNEL% is an integer value specifying the input channel to be read (0 - 31).

Returned Value:

VOLTAGE contains a single precision number that is the voltage read in microvolts or contains an error code:

-20,000,000 = channel number out of range
-30,000,000 = board number out of range
-40,000,000 = timeout error during conversion

Example:

The following BASIC program segment could be written to read an analog voltage from input channel 2 of the fourth card in the system.

```
40 CARDNO%=3
50 CHANNEL%=2
60 CALL AD1260AIN(VOLTAGE,
CARDNO%, CHANNEL%)
```

AD1260INIT

Structure:

CALL AD1260INIT(ARROR%,BASENO%, BASEADR%,NOCARDS%,INCHAN%,GAIN, ADCONF%)

Description:

This routine initializes the RTI-1260 library routines with the user-selected hardware configuration of the card if different from the default condition. The card default initialization is for one RTI-1260 card in a system with a base address of 130, gain equal 1V/1V, configured for 16 single-ended inputs of ±10V offset binary. There may be up to 22 of these cards in a system. When using more than one card, all cards with the same configuration are mapped contiguously and are initialized with the same routine call.

Input Parameters:

BASENO% is an integer value specifying the base number of the group of contiguous cards configured by this command (0 - 21).

BASEADR% is a hexadecimal integer value specifying the base address of the card. This is the lowest port address of the contiguously mapped RTI-1260 cards. The library routines use a 10-bit address. The RTI-1260 card should be mapped for an 8-bit I/O address. In software, set the most significant bits of the 10-bit address to 01 (for example, the RTI-1260 address 20 is entered as 120). The least significant four bits of the address must be 0. The routines do not work if the card is memory mapped.

NOCARDS% is an integer value specifying the number of contiguously mapped cards specified by this command (1 - 22).

INCHAN% is the input multiplexer configuration:

00 = single-ended
01 = differential
02 = pseudo-differential

GAIN is the voltage gain on the A/D input multiplied by a factor of 10,000 to enable 12-bits of accuracy, hence a gain 1.2346 is entered as 12,346. The allowable GAIN range is 10000 to 10000000, this corresponds to a gain of 1V/V to 1000V/V. The hardware gain is set via the gain resistor (R), the relationship between the resistor (R) and GAIN is
GAIN = 10000 * [(20k ohms / R) +1]

ADCONF% is the A/D input range and output code:

00 = 0V to +10V, straight binary
01 = -10V to +10V, offset binary
02 = -10V to +10V, two's complement

Returned Value:

ARROR% contains the returned integer error code:

0 = no error
-1 = board number out of range
-2 = invalid card count
(NOCARDS% >22)
-3 = invalid multiplexer configuration
-4 = invalid A/D input range/output code
-5 = timeout error during reset
-6 = gain out of range

Example:

The following BASIC program segment could be written to initialize the library for a group of four RTI-1260 cards after a group of three differently configured RTI-1260 cards have already been initialized. This group of cards is mapped contiguously beginning at 8-bit I/O address &H20. The cards are configured for differential inputs, gain equal 1V/V, 0 to +10V input range with straight binary code.

```
40 BASENO%=3
50 BASEADR%=&H120
60 NOCARDS%=4
70 INCHAN%=1
80 GAIN = 10000;
90 ADCONF%=0
100 CALL AD1260INIT(ARROR%,
BASENO%, BASEADR%,NOCARDS%,
INCHAN%, GAIN,ADCONF%)
```

Structure: **CALL AD1260RESET(ARROR%)**

Description: This routine software resets all the RTI-1260 cards in the system. The multiplexer is set to select input channel 0, a wait for any pending conversions to complete or a timeout is performed.

Input Parameters: None

Returned Value: **ARROR%** contains the returned integer error code:

0 = no error
-5 = timeout error during reset

Example: The following BASIC program segment resets all RTI-1260 cards in a system.

```
40 CALL AD1260RESET(ARROR%)
```

Structure:	CALL **AD1260TEST**(ARROR%,CARDNO%)
Description:	This routine self-tests an RTI-1260 card. A voltage is read from the A/D converter and self-test is passed if there is no timeout error.
Input Parameters:	**CARDNO%** is an integer value specifying the card to be tested (0 - 21).
Returned Value:	**ARROR%** contains the returned integer error code: 0 = no error -1 = invalid card selected -5 = timeout error
Example:	The following BASIC program segment tests the second RTI-1260 card in a system.

```
40 CARDNO%=1
50 CALL
AD1260TEST(ARROR%,CARDNO%)
```

Analog Devices
RTI-1262
Analog Output Card
BASIC library

The Analog Devices RTI-1262 Analog Output Card is a general-purpose, low-cost analog output card. There are four 12-bit D/A converters providing four output channels. Two of the four voltage outputs can be converted to a 4-20 milliamp current loop output by installing the correct hardware option.

The STD-LIB RTI-1262 software modules are as follows:

AD1262AMPS	Transforms the output current requested in the call from microamps to the nearest D/A code and writes an output current. The output current is in the range 4000 - 20000 microamps.
AD1262AOUT:	Transforms the output voltage requested in the call from millivolts to the nearest D/A code and writes an output voltage.
AD1262INIT:	Initializes the library software with the user-selected hardware configuration of the card.
AD1262RESET:	Performs a software reset of the card. The D/A input registers are set to output a voltage of 0V.

NOTE: The RTI-1262 as-shipped mapping is at hexadecimal base address FFF0 in the memory map. To be compatible with the library, the card needs to be port mapped with an 8-bit address. The default 8-bit base address for the library is hexadecimal C0 which results in the card using ports C0 through C7.

The Analog Devices convention is to define the base address of a card as an address with the last four bits being zero. The library has followed this convention. This means that an RTI-1262 card that has been remapped to the library default would be defined as a base address of 1C0 in the initialize command, the physical ports of the card are actually located at 1C0 through 1C7. When referring to contiguous mapping for RTI-1262 cards, the cards are mapped as though they occupied 8 ports per card.

To change an RTI-1262 card from its as-shipped mapping to the library default mapping, make the following changes to the jumpers:

remove 58-60, 55-56, 51-52
install 59-60, 56-57, 52-53, 9-10, 1-2

For technical questions concerning STD-LIB software for the Analog Devices RTI-1262, contact Pro-Log customer service. For technical questions concerning the operation of the Analog Devices RTI-1262 hardware, contact the appropriate Analog Devices customer service as follows:

STATE	PHONE NUMBER
Alaska	
Arizona	(714)-641-9391
California (southern)	Direct Sales Office
Colorado	
California (northern)	(408)-947-0633
Utah	Direct Sales Office
Delaware	(215)-643-7790
New Jersey	Direct Sales Office
Pennsylvania	
Maine	
Massachusetts	(617)-329-4700
New Hampshire	Factory
Rhode Island	
Illinois	
Indiana	
Iowa	
Kansas	
Kentucky	(312)-653-5000
Missouri	Direct Sales Office
North Dakota	
South Dakota	
Wisconsin	
All Other States	(617)-329-4700
	Factory

Structure:	CALL **AD1262AMPS(ARROR%,CARDNO%, CHANNEL%,AMPS%)**
Description:	This routine outputs a current to the selected channel. The selected current is input in microamps and converted to a binary program value for the V/I converter. The V/I converter must be installed for the channel and the channel must be configured for range 0 - 10 volts, straight binary encoding.
Input Parameters:	**CARDNO%** is an integer value specifying the card from which the current is to be output (0 - 21). **CHANNEL%** is an integer value specifying the output channel (0 - 3). **AMPS%** is an integer value to select the current to be output in microamps (4000 - 20000).
Returned Value:	**ARROR%** contains the error code: 0 = no error -1 = board number out of range -2 = channel number out of range -3 = current out of range

Example:

The following BASIC program segment could be written to output a current of 5000 microamps to channel 0 of the fourth card in the system.

```
40 CARDNO%=3
50 CHANNEL%=0
60 AMPS%=5000
100 CALL AD1262AMPS(ARROR%,
CARDNO%, CHANNEL%,AMPS%)
```

AD1262AOUT

Structure:

CALL **AD1262AOUT(ARROR%,CARDNO%, CHANNEL%,VOLTAGE%)**

Description:

This routine outputs a voltage to the selected channel. The selected voltage is input in millivolts and converted to a binary program value for the D/A converter based on the user-selected output configuration for the channel.

Input Parameters:

CARDNO% is an integer value specifying the card from which the voltage is to be output (0 - 21).

CHANNEL% is an integer value specifying the channel from which the voltage is to be output (0 - 3).

VOLTAGE% contains the analog voltage to be output in millivolts (-10000 to + 9990).

Returned Value:

ARROR% contains the returned integer error code:

0 = no error
-1 = channel number out of range
-2 = board number out of range
-3 = voltage parameter out of range

Example:

The following BASIC program segment could be written to output an analog voltage of 2V from output channel 1 of the fourth card in the system.

```
40 CARDNO%=3
50 CHANNEL%=1
60 VOLTAGE%=2000
70 CALL AD1262AOUT(ARROR%,
CARDNO%, CHANNEL%,VOLTAGE%)
```

Structure:

CALL AD1262INIT(ARROR%,BASENO%, BASEADR%,NOCARDS%,DA0CONF%, DA1CONF%,DA2CONF%,DA3CONF%)

Description:

This routine initializes the RTI-1262 library routines with the user-selected hardware configuration of the card if different from the default condition. The card default initialization is for one RTI-1262 card in a system with a base address of 1C0, configured for -10V to +10V offset binary encoding.

Input Parameters:

BASENO% is an integer value specifying the base number of the group of contiguous cards configured by this command (0 - 21).

BASEADR% is a hexadecimal integer value specifying the base address of the card. This is the lowest port address of the contiguously mapped RTI-1262 cards. The library routines use a 10-bit address. The RTI-1262 card should be mapped for an 8-bit I/O address. In software, set the most significant bits of the 10-bit address to 01 (for example, the RTI-1262 address 20 is entered as 120). The least significant four bits of the address must be 0. The routines do not work if the card is memory mapped.

NOCARDS% is an integer value specifying the number of contiguously mapped cards specified by this command (1 - 22).

DA0CONF% is the D/A channel 0 output range and input code:

00 = 0V to +5V, straight binary
01 = 0V to +10V, straight binary
02 = -5V to +5V, offset binary
03 = -10V to +10V, offset binary

DA1CONF% is the D/A channel 1 output range and input code:

00 = 0V to +5V, straight binary
01 = 0V to +10V, straight binary
02 = -5V to +5V, offset binary
03 = -10V to +10V, offset binary

DA2CONF% is the D/A channel 2 output range and input code:

00 = 0V to +5V, straight binary
01 = 0V to +10V, straight binary
02 = -5V to +5V, offset binary
03 = -10V to +10V, offset binary

DA3CONF% is the D/A channel 3 output range and input code:

00 = 0V to +5V, straight binary
01 = 0V to +10V, straight binary
02 = -5V to +5V, offset binary
03 = -10V to +10V, offset binary

Returned Value:

ARROR% contains the returned integer error code:

0 = no error
-1 = board number out of range
-2 = invalid card count
-3 = invalid D/A channel input code/output range

Example:

The following BASIC program segment could be written to initialize the library for a group of four RTI-1262 cards after a group of three differently configured RTI-1262 cards have already been initialized. This group of cards is mapped contiguously beginning at 8-bit I/O address 0x20. Output channels are configured for 0 to +10V output range with straight binary code.

```
40 BASENO%=3
50 BASEADR%=&H120
60 NOCARDS%=4
70 DA0CONF%=1
80 DA1CONF%=1
90 DA2CONF%=1
100 DA3CONF%=1
110 CALL AD1262INIT(ARROR%,
BASENO%, BASEADR%,NOCARDS%,
DA0CONF%, DA1CONF%,DA2CONF%,
DA3CONF%)
```

Structure:	CALL **AD1262RESET**(ARROR%)
Description:	This routine software resets all the RTI-1262 cards in the system. The D/A input registers are set to output a voltage of 0V. Note that the D/A output voltage may differ between hardware and software resets, depending on the user-selected hardware configuration.
Input Parameters:	None
Returned Value:	**ARROR%** contains the returned integer error code: 0 = no error
Example:	The following BASIC program segment resets all RTI-1262 cards in a system. 40 CALL AD1262RESET(ARROR%)

PRO-LOG
75XX,76XX,and 79XX
Input/Output Cards
BASIC library

The Pro-Log input/output cards are general-purpose, low-cost analog I/O cards. The PL7000 library routines are general-purpose I/O routines that can be used with the following Pro-Log cards: 7501, 7502, 7503, 7504, 7506, 7601A, 7602A, 7603A, 7604A, 7605, 7614, 7904, and 7914. Following the description of the STD-LIB routines is a description of each of the Pro-Log I/O cards that can be used with the library software. Shown below is a chart that shows which PL7000 routines can be used with each I/O card.

	7501	7502	7503	7504	7506	7601A	7602A	7603A	7604A	7605	7614	7904	7914
PL7000INIT	*	*	*	*	*	*	*	*	*	*	*	*	*
PL7000RDBIT			*		*	*		*	*	*	*	*	*
PL7000RDPORT			o		o	*		*	*	*	*	*	*
PL7000RESETPORT	*	*		*		*	*		*	*	*	*	*
PL7000WRTBIT	*	*		*		*	*		*	*	*	*	*
PL7000WRTBITRD									*	*	*	*	*
PL7000WRTPORT	o	o		o		*	*		*	*	*	*	*
PL7000WRTPORTRD									*	*	*	*	*

* = Routine can be used for card

o = Routine can be used for card, but other routine is typically preferred

The STD-LIB 7000 software modules are as follows:

PL7000INIT:	Initializes the library software with the user-selected hardware configuration of the card.
PL7000RDBIT:	Reads a bit from the selected port.
PL7000RDPORT:	Reads an 8-bit value from the selected port.
PL7000RESETPORT:	Software resets a group of contiguously mapped output ports by writing a 0 to all ports.
PL7000WRTBIT:	Writes a 1 or a 0 to the selected bit of a port.
PL7000WRTBITRD:	Writes a 1 or a 0 to the specified bit and port, then reads back the bit value to verify a proper write.
PL7000WRTPORT:	Writes a specified 8-bit pattern to a port.
PL7000WRTPORTRD:	Writes a specified 8-bit pattern to a port, then reads back an 8-bit value to verify a proper write.

Structure:	CALL **PL7000INIT(ARROR%,BASENO%, BASEADR%,NOCARDS%,NOPORTS%)**
Description:	This routine initializes the PL7000 library for port I/O operations with the user selected hardware configuration if different from the default condition. The default is for one card (card 0) in a system containing two contiguously mapped ports with the lowest port address at hexadecimal 040. When using more than one card, all cards with the same configuration are mapped contiguously and are initialized with the same routine call.
Input Parameters:	**BASENO%** is an integer value specifying the base number of the group of contiguous cards configured by this command (0 -21). **BASEADR%** is a hexadecimal integer value specifying the base address of the card. This is the lowest port address of the contiguously mapped PL7000 cards. **NOCARDS%** is an integer value specifying the number of contiguously mapped cards specified by this command (1 -22). **NOPORTS%** is an integer value specifying the number of ports on each card (1 - 8).

Returned Value:

ARROR% contains the returned integer error code:

0 = no error
-1 = board number out of range
-2 = invalid card count (number of cards > 22, or the number of cards causes the base address of the last card to be less than that of the first card (wraparound)).
-3 = invalid number of ports

Example:

The following BASIC program segment could initialize the library for a group of four 7501 cards after a group of three differently configured cards has already been initialized. This group of cards is mapped contiguously beginning at 8-bit I/O address &H20. The cards are configured for 2 ports per card.

```
40 BASENO%=3
50 BASEADR%=&H120
60 NOCARDS%=4
70 NOPORTS%=2
80 CALL PL7000INIT(ARROR%,
BASENO%, BASEADR%,NOCARDS%,
NOPORTS%)
```

Structure:	CALL **PL7000RDBIT(CARDNO%, PORTNO%, BITNO%)**
Description:	This routine reads a bit from a port and returns 0, bit cleared, or 1 bit set.
Input Parameters:	**CARDNO%** is an integer value specifying the card to read (0 - 21). **PORTNO%** is an integer value specifying the port to be read (0 - 7). **BITNO%** is an integer value specifying the bit to be read (0 - 7).
Returned Value:	**ARROR%** contains the returned bit value or integer error code: 1 = bit is set 0 = bit is cleared -1 = card number out of range -2 = port number out of range -3 = bit number out of range
Example:	The following BASIC program segment could be written to read bit four of the second port and the second card in the system.

```
40 CARDNO%=1
50 PORTNO%=1
60 BITNO%=4
70 CALL PL7000RDBIT(ARROR%,
CARDNO%, PORTNO%,BITNO%)
```

PL7000RDPORT

Structure:	CALL **PL7000RDPORT(ARROR%, CARDNO%,PORTNO%)**
Description:	This routine reads an 8-bit value from a port.
Input Parameters:	**CARDNO%** is an integer value specifying the card to be read from (0 - 21). **PORTNO%** is an integer value specifying the port to be read (0 - 7).
Returned Value:	**ARROR%** contains the returned port value (0-255) or an integer error code: -1 = card number out of range -2 = port number out of range
Example:	The following BASIC program segment could be written to read port 4 of the middle card in a three-card system.

```
40 CARDNO%=1
50 PORTNO%=4
60 CALL PL7000RDPORT(ARROR%,
CARDNO%,PORTNO%)
```

Structure:	CALL **PL7000RESETPORT(ARROR%, BASENO%)**
Description:	This routine software resets a group of contiguously mapped output ports. This group of cards was defined by PL7000INIT. All ports in this group of contiguous cards should be output ports or input/output ports. A zero is written to all the ports.
Input Parameters:	**BASENO%** is the base number of the group of contiguous cards reset by this command (0 - 21).
Returned Value:	**ARROR%** contains the returned integer error code: 0 = no error -1 = base number out of range
Example:	The following BASIC program segment resets a group of contiguously mapped cards beginning at card 0.

```
40 BASENO%=0
50 CALL PL7000RESETPORT(ARROR%,
BASENO%)
```

PL7000

Structure:	**CALL PL7000WRTBIT(ARROR%, CARDNO%,PORTNO%,BIT%,BITNO%)**
Description:	This routine writes a 1 or a 0 to a specified bit of a port.
Input Parameters:	**CARDNO%** is an integer value specifying the card to be written to (0 - 21).
	PORTNO% is an integer value specifying the port to be written to (0 - 7).
	BIT% is the data to be written to the bit (0 or 1).
	BITNO% is an integer value specifying the bit of the port being written to (0 - 7).
Returned Value:	**ARROR%** contains the returned integer error code: 0 = no error -1 = invalid card selected -2 = port number out of range -3 = bit number out of range -4 = invalid write data

Example:

The following BASIC program segment sets bit 0 of the second port on card 2 of the system.

```
40 CARDNO%=2
50 PORTNO%=1
60 BIT%=1
70 BITNO%=0
80 CALL PL7000WRTBIT(ARROR%,
CARDNO%,PORTNO%,BIT%,BITNO%)
```

Structure:	CALL **PL7000WRTBITRD**(ARROR%, CARDNO%,PORTNO%,BIT%,BITNO%)
Description:	This routine writes a 1 or a 0 to a specified bit of a port, then reads back the bit value to verify a proper write. This routine can only be used by cards with the readback capability.
Input Parameters:	**CARDNO%** is an integer value specifying the card to be written to (0 - 21). **PORTNO%** is an integer value specifying the port to be written to (0 - 7). **BIT%** is the data to be written to the bit (0 or 1). **BITNO%** is an integer value specifying the bit of the port being written to (0 - 7).
Returned Value:	**ARROR%** contains the returned integer error code: 0 = no error -1 = invalid card selected -2 = port number out of range -3 = bit number out of range -4 = invalid write data -5 = read back error

Example:

The following BASIC program segment sets bit 0 of the second port on card 2 of the system.

```
40 CARDNO%=2
50 PORTNO%=1
60 BIT%=1
70 BITNO%=0
80 CALL PL7000WRTBITRD(ARROR%,
CARDNO%,PORTNO%,BIT%,BITNO%)
```

PL7000WRTPORT

Structure:	CALL **PL7000WRTPORT(ARROR%, CARDNO%,PORTNO%,PORTDATA%)**
Description:	This routine writes an 8-bit pattern to a specified port.
Input Parameters:	**CARDNO%** is an integer value specifying the card to be written to (0 - 21). **PORTNO%** is an integer value specifying the port to be written to (0 - 7). **PORTDATA%** is the data to be written to the port (0 hex - FF hex).
Returned Value:	**ARROR%** contains the returned integer error code: 0 = no error -1 = invalid card selected -2 = port number out of range

Example:

The following BASIC program segment writes 4B hex to port 0 of the second card of the system.

```
40 CARDNO%=1
50 PORTNO%=0
60 PORTDATA%=&H4B
70 CALL PL7000WRTPORT(ARROR%,
CARDNO%,PORTNO%,PORTDATA%)
```

Structure:	CALL **PL7000WRTPORTRD(ARROR%, CARDNO%,PORTNO%,PORTDATA%)**
Description:	This routine writes an 8-bit pattern to a specified port, then reads back an 8-bit value to verify a proper write. This routine can only be used by cards with the readback capability.
Input Parameters:	**CARDNO%** is an integer value specifying the card to be written to (0 - 21). **PORTNO%** is an integer value specifying the port to be written to (0 - 7). **PORTDATA%** is the data to be written to the port (0 hex - FF hex).
Returned Value:	**ARROR%** contains the returned integer error code: 0 = no error -1 = invalid card selected -2 = port number out of range -3 = readback error

Example:

The following BASIC program segment writes 4B hex to port 0 of the second card of the system.

```
40 CARDNO%=1
50 PORTNO%=0
60 PORTDATA%=&H4B
70 CALL PL7000WRTPORTRD(ARROR%,
CARDNO%,PORTNO%,PORTDATA%)
```

The following Pro-Log I/O cards may be used with the STD-LIB software:

7501 DC DRIVER CARD

The 7501 converts TTL level signals supplied by the STD BUS to latched, negative true, open-collector DC drive signals. The 7501 provides 16 independent DC output circuits. Each open collector output circuit is capable of sinking up to 225 mA of current in the on state and can withstand a 50V output level in the off state. The two output ports are cleared by SYSRESET* and are consecutively mapped at addresses 40 and 41 as shipped.

The 7501 uses the general-purpose I/O routines of STD-LIB as follows:

PL7000INIT:	Initializes the library software with the user-selected hardware configuration.
PL7000RESETPORT:	Software resets the output ports.
PL7000WRTBIT:	Writes a 1 or a 0 to a specified bit of a port.

7502 SPST RELAY OUTPUT CARD

The 7502 consists of eight independent SPST dry reed relays controlled by a fully decoded, latched output port. The output port is cleared by SYSRESET* and is mapped at address 40 as shipped.

The 7502 uses the general-purpose I/O routines of STD-LIB as follows:

PL7000INIT:	Initializes the library software with the user-selected hardware configuration.
PL7000RESETPORT:	Software resets the output ports.
PL7000WRTBIT:	Writes a 1 or a 0 to a specified bit of the port.

7503 OPTOISOLATED INPUT CARD

The 7503 converts an AC or DC input voltage into a TTL-level signal that is read by an on-card STD BUS input port. The 7503 provides eight independent AC/DC inputs. An optical coupling circuit isolates the circuits from each other and from the STD BUS system. The 7503 offers a choice of two input ranges. Range selection is independent for each input and is selected through the installation of wire jumpers. The low range responds to input voltages between 4.5 VRMS and 22 VRMS for 5V, 6V, and 12V applications. The high range responds to input voltages of between 20 VRMS and 80 VRMS and is useful for systems using 24V, 28V, and 48V. The input port is mapped at address 40 as shipped.

The 7503 uses the general-purpose I/O routines of STD-LIB as follows:

PL7000INIT:	Initializes the library software with the user-selected hardware configuration.
PL7000RDBIT:	Reads a bit from the port.

7504 TRIAC OUTPUT CARD

The 7504 consists of eight independent solid-state AC relays (triacs) controlled by a fully decoded, latched 8-bit output port. The 7504 allows the processor direct control of eight switched AC power circuits (40 - 280 VRMS and 47 - 63 Hz). The output port is cleared by SYSRESET* and is mapped at address 40 as shipped.

The 7504 uses the general-purpose I/O routines of STD-LIB as follows:

PL7000INIT:	Initializes the library software with the user-selected hardware configuration.
PL7000RESETPORT:	Software resets the output ports.
PL7000WRTBIT:	Writes a 1 or a 0 to a specified bit of the port.

7506 OPTOISOLATED INPUT CARD

The 7506 converts an AC or DC input voltage into a TTL-level signal that is read by an on-card STD BUS input port. The 7506 provides eight independent AC/DC inputs. An optical coupling circuit isolates the circuits from each other and from the STD BUS system. The 7506 offers a choice of two input ranges. Range selection is independent for each input and is selected through the installation of wire jumpers. The low range responds to input voltages between 70 VRMS and 150 VRMS for 120V applications. The high range responds to input voltages of between 140 VRMS and 280 VRMS and is useful for systems using 240V. The input port is mapped at address 40 as shipped.

The 7506 uses the general-purpose I/O routines of STD-LIB as follows:

PL7000INIT:	Initializes the library software with the user-selected hardware configuration.
PL7000RDBIT:	Reads a bit from the port.

7601A TTL INPUT/OUTPUT PORT CARD

The 7601A provides four 8-bit gated input ports (32 input lines) and four 8-bit latched output ports (32 output lines). The four consecutive independent input ports and output ports are mapped at address 00 - 03 as shipped. The output ports are cleared by SYSRESET*.

The 7601A uses the general-purpose I/O routines of STD-LIB as follows:

PL7000INIT:	Initializes the library software with the user-selected hardware configuration.
PL7000RDBIT:	Reads a bit from a port.
PL7000RDPORT:	Reads an 8-bit value from a port.
PL7000RESETPORT:	Software resets the output ports.
PL7000WRTBIT:	Writes a 1 or a 0 to a specified bit of a port.
PL7000WRTPORT:	Writes an 8-bit pattern to a port.

7602A TTL OUTPUT PORT CARD

The 7602A provides eight 8-bit latched output ports (64 lines). The consecutively mapped ports are cleared by SYSRESET* and are mapped at address 00 -07 as shipped.

The 7602A uses the general-purpose I/O routines of STD-LIB as follows:

PL7000INIT:	Initializes the library software with the user-selected hardware configuration.
PL7000RESETPORT:	Software resets the output ports.
PL7000WRTBIT:	Writes a 1 or a 0 to a specified bit of a port.
PL7000WRTPORT:	Writes an 8-bit pattern to a port.

7603A TTL INPUT PORT CARD

The 7603A provides eight 8-bit gated input ports (64 input lines). The consecutive ports are mapped at address 00 - 07 as shipped.

The 7603A uses the general-purpose I/O routines of STD-LIB as follows:

PL7000INIT:	Initializes the library software with the user-selected hardware configuration.
PL7000RDBIT:	Reads a bit from a port.
PL7000RDPORT:	Reads an 8-bit value from a port.

7604A TTL INPUT/OUTPUT CARD

The 7604A provides up to eight ports of TTL I/O. The eight ports can be individually configured as input, output, or output with readback. The consecutive ports are mapped at address 00 - 07 as shipped. The output ports are cleared by SYSRESET*.

The 7604A uses the general-purpose I/O routines of STD-LIB as follows:

PL7000INIT:	Initializes the library software with the user-selected hardware configuration.
PL7000RDBIT:	Reads a bit from a port.
PL7000RDPORT:	Reads an 8-bit value from a port.
PL7000RESETPORT:	Software resets the output ports.
PL7000WRTBIT:	Writes a 1 or a 0 to a specified bit of a port.
PL7000WRTBITRD:	Writes a 1 or a 0 to a specified bit of a port and then reads back the value to verify a proper write.
PL7000WRTPORT:	Writes an 8-bit pattern to a port.
PL7000WRTPORTRD:	Writes an 8-bit pattern to a port and then reads back the value to verify a proper write.

7605 PROGRAMMABLE TTL I/O CARD

The 7605 is a general-purpose TTL I/O card that provides four 8-bit I/O ports. Since each of the 32 I/O lines can be individually programmed as input, output, or output with readback, the 7605 is particularly useful when an external device requires more or less than eight lines or a mixture of inputs and outputs. The four consecutive ports are mapped at address 00 - 03 as shipped. The output ports are cleared by SYSRESET*.

The 7605 uses the general-purpose I/O routines of STD-LIB as follows:

PL7000INIT:	Initializes the library software with the user-selected hardware configuration.
PL7000RDBIT:	Reads a bit from a port.
PL7000RDPORT:	Reads an 8-bit value from a port.
PL7000RESETPORT:	Software resets the output ports.
PL7000WRTBIT:	Writes a 1 or a 0 to a specified bit of a port.
PL7000WRTBITRD:	Writes a 1 or a 0 to a specified bit of a port and then reads back the value to verify a proper write.
PL7000WRTPORT:	Writes an 8-bit pattern to a port.
PL7000WRTPORTRD:	Writes an 8-bit pattern to a port and then reads back the value to verify a proper write.

7614 64-LINE TTL I/O CARD

The 7614 provides eight 8-bit I/O ports, 64 I/O lines total. Each port can be configured as input, output, or output with readback. The 7614 operates in systems using an 8- or 10-bit I/O address. The I/O ports can be configured as four or eight ports, and are mapped at address 00 - 07 as shipped. The output ports are cleared by SYSRESET*.

The 7614 uses the general-purpose I/O routines of STD-LIB as follows:

PL7000INIT:	Initializes the library software with the user-selected hardware configuration.
PL7000RDBIT:	Reads a bit from a port.
PL7000RDPORT:	Reads an 8-bit value from a port.
PL7000RESETPORT:	Software resets the output ports.
PL7000WRTBIT:	Writes a 1 or a 0 to a specified bit of a port.
PL7000WRTBITRD:	Writes a 1 or a 0 to a specified bit of a port and then reads back the value to verify a proper write.
PL7000WRTPORT:	Writes an 8-bit pattern to a port.
PL7000WRTPORTRD:	Writes an 8-bit pattern to a port and then reads back the value to verify a proper write.

7904 DECODED I/O UTILITY CARD

The 7904 is a printed circuit card designed for prototyping I/O circuitry. It provides complete STD BUS buffering and decoding for I/O cards with input and output port strobe logic and access to memory page and CPU/memory control lines. The card is configured for eight consecutive I/O ports mapped at address 00 - 07 as shipped.

The 7904 can use the appropriate general-purpose I/O routines of STD-LIB as follows:

PL7000INIT:	Initializes the library software with the user-selected hardware configuration.
PL7000RDBIT:	Reads a bit from a port.
PL7000RDPORT:	Reads an 8-bit value from a port.
PL7000RESETPORT:	Software resets the output ports.
PL7000WRTBIT:	Writes a 1 or a 0 to a specified bit of a port.
PL7000WRTBITRD:	Writes a 1 or a 0 to a specified bit of a port and then reads back the value to verify a proper write.
PL7000WRTPORT:	Writes an 8-bit pattern to a port.
PL7000WRTPORTRD:	Writes an 8-bit pattern to a port and then reads back the value to verify a proper write.

7914 DECODED I/O UTILITY CARD

The 7914 is a printed circuit card designed for prototyping I/O circuitry. The card provides STD BUS buffering and I/O port select decoding. The card is configured for eight consecutive I/O ports mapped at address 0000 - 0007 as shipped.

The 7914 can use the appropriate general-purpose I/O routines of STD-LIB as follows:

PL7000INIT:	Initializes the library software with the user-selected hardware configuration.
PL7000RDBIT:	Reads a bit from a port.
PL7000RDPORT:	Reads an 8-bit value from a port.
PL7000RESETPORT:	Software resets the output ports.
PL7000WRTBIT:	Writes a 1 or a 0 to a specified bit of a port.
PL7000WRTBITRD:	Writes a 1 or a 0 to a specified bit of a port and then reads back the value to verify a proper write.
PL7000WRTPORT:	Writes an 8-bit pattern to a port.
PL7000WRTPORTRD:	Writes an 8-bit pattern to a port and then reads back the value to verify a proper write.

The 7303 Keyboard Display Card is a general-purpose control panel card with data input and display capability. The card includes an 8-character alphanumeric display, keyboard with 24 program-definable keys plus system reset, an 8-bit binary LED display, and two rocker switches. The alphanumeric displays do not display lower case characters. The display routines convert characters to upper case.

The STD-LIB 7303 software modules are as follows:

PL7303CHRSHFT:	Shifts the displayed characters to the left and brings in a new character at the right.
PL7303CHRWR:	Writes an ASCII character to a specified display position.
PL7303CURSOR:	Sets or removes one or more cursors from the display.
PL7303GETKEY:	Waits for and then reads a keystroke. The routine waits for a new keystroke and then waits for key debounce.
PL7303GETROCKR:	Reads a valid rocker switch state. The routine waits for any switch bouncing to finish.
PL7303INIT:	Initializes the library software with the user-selected hardware configuration of the card. This routine also performs self-test functions: a write to the displays and LEDs for visual check, and a keyboard check for a "stuck key."

PL7303KEYFETCH:	To be called 15 ms after the PL7303KEYSTAT routine. It returns a key if bouncing has ended and a new keystroke has been detected.
PL7303KEYSTAT:	Checks to see if a new key is currently closed. The routine does not indicate whether the depressed key has stopped bouncing.
PL7303MESSAGE:	Shifts a message into the display from the right and continues shifting the message across until the display is blank.
PL7303RCKRSTAT:	Returns the rocker switch status: open, closed, or changed.
PL7303RESET:	Blanks all displays.
PL7303SETLEDS:	Sets the state of the LEDs.
PL7303TEST:	Performs a self-test of the card, including a write to the displays and LEDs for visual check, and a keyboard check for a "stuck key."
PL7303WAIT:	Waits 15 ms for debouncing a key in the keyboard or a rocker switch.
PL7303WORD:	Displays an 8-character (or shorter) word from left to right and blank fills any remaining portion of the display.

PL7303CHRSHFT

Structure: CALL PL7303CHRSHFT(ARROR%, CHAR%)

Description: This routine shifts the displayed characters to the left and brings in a new character at the right.

Input Parameters: CHAR% is an integer number whose value is the ASCII code for the character to be shifted into the display at the right.

Returned Value: ARROR% contains the returned integer error code:

0 = no error
-1 = no 7303 installed
-2= invalid character selected

Example: The following BASIC program segment could be written to shift the display left and bring in an "A" to the rightmost display.

```
40  CHAR%=ASC("A")
50  CALL PL7303CHRSHFT(
ARROR%,CHAR%)
```

PL7303CHRWR

Structure: **CALL PL7303CHRWR(ARROR%, CHAR$,DISPNO%)**

Description: This routine writes an ASCII character to the specified display.

Input Parameters: **CHAR$** is a one-character ASCII string to be written to a display.

DISPNO% is an integer value specifying the display to be written (0 - 7). The rightmost display is defined as display 0.

Returned Value: **ARROR%** contains the returned integer error code:

0 = no error
-1 = no 7303 installed
-2 = invalid display selected
-3 = invalid character selected

Example: The following BASIC program segment could be written to write an "A" to the third display from the left.

```
40  CHAR$="A"
50  DISPNO%=5
60  CALL PL7303CHRWR(ARROR%,
CHAR$,DISPNO%)
```

PL7303CURSOR

Structure:

CALL PL7303CURSOR(ARROR%, DISPLAY%,ONOFF%)

Description:

This routine writes a cursor (all segments lit) to the specified displays. With a valid ASCII character loaded to a display position, the cursor can also be displayed in that position. When the cursor is removed, the ASCII character reappears.

Input Parameters:

DISPLAY% is a binary integer indicating which displays are to be written to.

ONOFF% is a binary integer value specifying whether the associated cursor is to be turned on or off (1 = on, 0 = off).

Returned Value:

ARROR% contains the returned integer error code:

0 = no error
-1 = no 7303 installed

Example:

The following BASIC program segment could be written to write a cursor to the third display from the left and to turn off the rightmost cursor.

```
40  DISPLAY%=&H21
50  ONOFF%=&H20
60  CALL PL7303CURSOR(ARROR%,
DISPLAY%,ONOFF%)
```

Structure:	**CALL PL7303GETKEY(KEY%)**
Description:	This routine waits for a keystroke and then waits for key debounce. The routine then returns with the key that was pressed.
Input Parameters:	None
Returned Value:	**KEY%** contains the key that was detected or an error code: &H0-&H17 = key detected -1 = no 7303 installed
Example:	The following BASIC program segment could be written to read a key from the keypad.

```
40  CALL PL7303GETKEY(KEY%)
```

Structure:	**CALL PL7303GETROCKR(STAT%)**
Description:	This routine reads a valid rocker switch state. The routine waits for any switch bouncing to end.
Input Parameters:	None
Returned Value:	**STAT%** contains the returned rocker switch status: 0 = both switches open 1 = SW1 closed, SW2 open 2 = SW2 closed, SW1 open 3 = both switches closed -1 = no 7303 installed
Example:	The following BASIC program segment could be written to get the valid status of rocker switch 2.

```
40  CALL PL7303GETROCKR( STAT%)
```

PL7303INIT

Structure:	**CALL PL7303INIT(ARROR%, BASEADR%)**
Description:	This routine initializes the 7303 library routines with the configuration of the card if different from the as-shipped condition (BASEADR% = &HD0). The storage for the previous condition of the rocker switches is initialized to open. There may be only one 7303 card in a system. This routine also lights all LEDs and all the segments on the displays for a one-second visual check of proper operation. A "stuck key" test is performed, in which the keyboard is checked to see that no key is depressed. All LEDs and displays are then blanked.
Input Parameters:	**BASEADR%** is a hexadecimal integer value specifying the base address of the card. This is the lowest port address of the 7303 card. The library routines use a 10-bit address, even though the card can only be set to an 8-bit address. Set the two most significant bits of the 10-bit address to 01 (for example, 7303 address D0 is entered as 1D0).
Returned Value:	**ARROR%** contains the returned integer error code: 0 = no error -1 = no 7303 installed other = row of stuck key (binary coded) bit 0=row1:keys 0-3 bit 1=row2:keys 4-7 bit 2=row3:keys 8-B bit 3=row4:keys C-F bit 4=row5:keys 10-13

bit 5=row6:keys 14-17

Example:

The following BASIC program segment could be written to initialize a 7303 card mapped at &HC0.

```
40  BASEADR%=&H1C0%
50  CALL PL7303INIT(ARROR%,
BASEADR%)
```

Structure:	**CALL PL7303KEYFETCH(STAT%)**
Description:	This routine reads the status of the keypad. It is called 15 ms after the PL7303KEYSTAT routine.
Input Parameters:	None
Returned Value:	**STAT%** contains the returned keypad status. &H0 - &H17 for the key number if a valid key depression was detected. -1 = no 7303 installed -2 = no valid key detected
Example:	The following BASIC program segment could be written to read the key status (typically written after PL7303KEYSTAT and PL7303WAIT routines or their equivalents have been called).

```
40  CALL PL7303KEYFETCH(STAT%)
```

Structure:	**CALL PL7303KEYSTAT(STAT%)**
Description:	This routine checks to see if a new key on the keypad is currently closed. The routine does not indicate whether the depressed key has stopped bouncing.
Input Parameters:	None
Returned Value:	**STAT%** contains the returned keypad status: 0 = no new key closed 1 = new key closed -1 = no 7303 installed
Example:	The following BASIC program segment could be written to get the status of the keypad.

```
40  CALL PL7303KEYSTAT( STAT%)
```

PL7303MESSAGE

Structure: **CALL PL7303MESSAGE(ARROR%, MSG$)**

Description: This routine writes an ASCII message to the displays from the right. The message is shifted through the displays until all the displays are left blank. Any non-ASCII characters in the string are replaced with a space.

Input Parameters: **MSG$** is the string value to be displayed. The string must have between 1 and 65535 characters.

Returned Value: **ARROR%** contains the returned integer error code:

 0 = no error
-1 = no 7303 installed

Example: The following BASIC program segment could be written to scroll a message through the displays.

```
40  MSG$="PRINT DEMONSTRATION"
50  CALL PL7303MESSAGE(ARROR%,
MSG$)
```

Structure:	**CALL PL7303RCKRSTAT(STAT%)**
Description:	This routine reads the current status of a rocker switch (open, closed, or changed). The routine does not indicate whether a rocker that has changed status since the last call has stopped bouncing.
Input Parameters:	None
Returned Value:	**STAT%** contains the returned rocker switch status: 1 = SW1 status change 2 = SW2 status change 3 = SW1 and SW2 status change 4 = no status change, both switches open 5 = no status change, SW1 closed, SW2 open 6 = no status change, SW2 closed, SW1 open 7 = no status change, both switches closed -1 = no 7303 installed
Example:	The following BASIC program segment could be written to get the status of the rocker switches.

```
40  CALL PL7303RCKRSTAT(STAT%)
```

PL7303RESET

Structure: **CALL PL7303RESET(ARROR%)**

Description: This routine software resets the 7303. All alphanumeric and LED displays are blanked.

Input Parameters: None

Returned Value: **ARROR%** contains the returned integer error code:

 0 = no error
-1 = no 7303 installed

Example: The following BASIC program segment resets the 7303.

```
40  CALL PL7303RESET(ARROR%)
```

Structure:

CALL PL7303SETLEDS(ARROR%, LEDS%,ONOFF%)

Description:

This routine sets the state of the LED displays. Note that many of the other 7303 routines can affect the state of the LEDs. The LEDs are restored to their previous condition after any of these routines are called.

Input Parameters:

LEDS% is a binary integer number indicating which LEDs are to be changed.

ONOFF% is a binary integer number specifying the values for the LEDs that are being written (1 = on, 0 = off).

Returned Value:

ARROR% contains the returned integer error code:

 0 = no error
-1 = no 7303 installed

Example:

The following BASIC program segment could be written to set the third LED from the left and clear the fifth LED from the left.

```
40  LEDS%=&H28
50  ONOFF%=&H20
60  CALL PL7303SETLEDS(ARROR%,
LEDS%,ONOFF%)
```

PL7303TEST

Structure: **CALL PL7303TEST(ARROR%)**

Description: This routine performs a self-test of the 7303. It lights all LEDs and all the segments on the displays for a one-second visual check of proper operation. A "stuck key" test is performed, in which the keyboard is checked to see that no key is depressed. All LEDs and displays are then blanked.

Input Parameters: None

Returned Value: ARROR% contains the returned integer error code:

0 = no error
-1 = no 7303 installed
other = row of stuck key
    (binary coded)
    bit 0=row1:keys 0-3
    bit 1=row2:keys 4-7
    bit 2=row3:keys 8-B
    bit 3=row4:keys C-F
    bit 4=row5:keys 10-13
    bit 5=row6:keys 14-17

Example: The following BASIC program segment could be written to self-test a 7303 card.

```
40  CALL PL7303TEST(ARROR%)
```

Structure:	**CALL PL7303WAIT(ARROR%)**
Description:	This routine waits 15 ms, the time required to debounce a keyboard switch or rocker switch.
Input Parameters:	None
Returned Value:	ARROR% contains the returned integer error code: 0 = no error -1 = no 7303 installed
Example:	The following BASIC program segment waits 15 ms for a 7303 switch to debounce.

```
40  CALL PL7303WAIT(ARROR%)
```

PL7303WORD

Structure:

CALL PL7303WORD(ARROR%,MSG$)

Description:

This routine writes an ASCII 8-character (or shorter) word to the display from left to right. Blanks are written to any remaining portion of the display.

Input Parameters:

MSG$ is the string value to be displayed.

Returned Value:

ARROR% contains the returned integer error code:

 0 = no error
-1 = no 7303 installed

Example:

The following BASIC program segment could be written to write "TEST" to the first four display positions, followed by four blanks.

```
40  MSG$="TEST"
50  CALL PL7303WORD(ARROR%,
MSG$)
```

PRO-LOG
7312/7314A
Communications Card
BASIC library

The 7312 and 7314A Peripheral Interface Cards provide full-duplex RS-232-C serial channels for connection to terminals, printers, modems, and other peripherals or systems. Each card features two (7312) or four (7314A) serial channels, each with selectable DCE/DTE configuration using standard RS-232-C line drivers and receivers.

Up to 32 RS-232-C channels are supported by the STD-LIB software modules. Each channel can be separately configured for baud rate, data bits, stop bits, handshake, and parity. Input and output are interrupt driven with user-defined receive and transmit buffers.

The STD-LIB 7312 and 7314A software modules are as follows:

PL7314BUFFLUSH:	Removes all characters from the transmit and/or receive buffer.
PL7314CHANRST:	Initializes one RS-232-C channel.
PL7314INIT:	Initializes the library software with the user-selected hardware configuration for all channels.
PL7314INTEN:	Enables or disables receive, transmit, or break interrupts.
PL7314RCV:	Reads one received character from the receive buffer.
PL7314RCVCHK:	Checks to see if a received character is available in the receive buffer.

PL7314SETBUF:	Allocates a transmit or receive buffer to be used in place of the default 8-character buffers allocated by PL7314INIT.
PL7314TEST:	Tests a serial channel by writing to and reading from the control registers for the channel.
PL7314XMT:	Transmits a character by placing it in the transmit buffer.
PL7314XMTBRK:	Transmits a break condition immediately.
PL7314XMTCHK:	Checks to see if there is room in the transmit buffer for a character.
PL7314XMTIMD:	Transmits one character immediately without using the transmit buffer.

NOTE: The 7312 and 7314A as-shipped mapping is at hexadecimal address 0B0 through 0B7. When one of these cards is installed in an STD-DOS system with the as-shipped mapping, STD-DOS initializes the card as communication ports COM1 through COM4. If the card is not used for STD-DOS communication ports, the library can simply reinitialize the card. To avoid the possibility of someone attempting to use a 7312 or 7314A card for two different purposes simultaneously, the default library address for a 7312 or 7314A card is different from the address used by STD-DOS. The library default addresss is hexadecimal 160 through 167.

To change a 7312-1 or 7314A-1 card from its as-shipped mapping to the library default mapping, make the following changes to the W1 jumper header:

remove 5-6
install 7-8
remove 9-10
install 13-14

Structure:	CALL PL7314BUFFLUSH(ARROR%, CHAN%,XMT%,RCV%)
Description:	This routine removes all characters from the transmit and/or receive buffers.
Input Parameters:	CHAN% is an integer value specifying the number of the channel for which the buffer(s) is being flushed (0-31). XMT% is an integer value specifying whether the transmit buffer is to be flushed (XMT%=1) or not (XMT%=0). RCV% is an integer value specifying whether the receive buffer is to be flushed (RCV%=1) or not (RCV%=0).
Returned Value:	ARROR% contains the returned integer error code: 0 = no error -1 = invalid channel selected
Example:	The following BASIC program segment could be written to flush the transmit buffer for channel 4.

```
40  CHAN%=4
50  XMT%=1
60  RCV%=0
70  CALL PL7314BUFFLUSH(
ARROR%,CHAN%,XMT%,RCV%)
```

PL7314CHANRST

Structure:

**CALL PL7314CHANRST(ARROR%,
CHAN%,BAUD%,PARITY%,
BITSCHAR%,STOPBIT%,HANDSHK%)**

Description:

This routine initializes a channel and enables the transmit, receive, and break interrupts for one channel of a 7312 or 7314A card. This routine must be called once for each serial channel in the system. The PL7314INIT routine must be called before this routine can be called. The default 8-character transmit and receive buffers are allocated within the PL7314INIT module. Different size buffers can be allocated by the PL7314SETBUF module.

Input Parameters:

CHAN% is an integer value specifying the number of the channel being initialized (0-31).

BAUD% is an integer value specifying the baud rate (19200, 9600, 4800, 2400, 1200, 600, 300, 150, 110).

PARITY% is an integer value specifying parity (0 = no parity, 1 = odd parity, 2 = even parity).

BITSCHAR% is an integer value specifying the number of data bits per character (5, 6, 7, 8).

STOPBIT% is an integer value specifying the number of stop bits (1 or 2).

HANDSHK% is an integer value specifying the handshake (0 = no handshake, 1 = CTS, 2 = XON/XOFF).

Returned Value:

ARROR% contains the returned integer error code:

0 = no error
-1 = error, channel cannot be
initialized as requested

Example:

The following BASIC program segment could be written to initialize serial channel 1 of a system. The channel is initialized to 9600 baud, no parity, 8 data bits per character, one stop bit, and XON/XOFF handshake protocol.

```
40  CHAN%=1
50  BAUD%=9600
60  PARITY%=0
70  BITSCHAR%=8
80  STOPBIT%=1
90  HANDSHK%=2
100 CALL PL7314CHANRST(
ARROR%,CHAN%,BAUD%,PARITY%,BIT
SCHAR%,STOPBIT%,HANDSHK%)
```

Structure:	DIM BUFF%(MAXCHAN%*24) **CALL PL7314INIT(ARROR%, BASEADR%,MAXCHAN%, VECTLO%,BUFF%(0))**
Description:	This routine initializes the 7312 or 7314A library routines with the user-selected hardware configuration for all serial channels. All 7312 or 7314A cards must be mapped contiguously. This routine checks that all indicated cards are properly installed and mapped. This routine must be called before any other 7314 module is called.
Input Parameters:	**BASEADR%** is a hexadecimal integer value specifying the 10-bit base address of the cards. This is the lowest port address of the contiguously mapped 7312 and 7314A cards. **MAXCHAN%** is an integer value specifying the number of RS-232-C channels supported by 7312/7314A routines (1 - 32). **VECTLO%** is a hexadecimal integer value specifying the first interrupt vector available for receive and transmit interrupts. Starting with this interrupt, this software initializes and uses n consecutive interrupts, where n=MAXCHAN%/2 (each pair of channels uses one interrupt). **BUFF%**(0) is the first element of a buffer that can be used by the 7312/7314A software. The size of the buffer in words must be MAXCHAN% times 24 (24 words per channel are required).
Returned	**ARROR%** contains the returned

Value: integer error code:

0 = no error
n = number of requested channels
that could not be initialized
(n is a number from 1 to 32)

Example: The following BASIC program segment could be written to initialize the library for ten RS-232-C channels, located on two 7314A cards and one 7312 card. Channels 0-3 are located on a 7314A mapped at &H100-&H107, channels 4-7 on a 7314A mapped at &H108-&H10F, and channels 8 and 9 on a 7312 mapped at &H110-&H113. Interrupt vectors C0-C4 are available for use by this module. Memory location BUFADR contains a doubleword pointer to a 240-word RAM buffer.

```
50  DIM BUFF%(240)
60  BASEADR%=&H100
70  MAXCHAN%=10
80  VECTLO%=&HCO
90  CALL PL7314INIT(ARROR%,
BASEADR%,MAXCHAN%,VECTLO%,
BUFF%(0))
```

PL7314INTEN

Structure:

CALL PL7314INTEN(ARROR%, XMTINT%,RCVINT%,BRKINT%, CHAN%)

Description:

This routine enables or disables receive, transmit, and break interrupts for a channel.

Input Parameters:

XMTINT% is an integer value specifying whether the transmitter interrupt for the channel is enabled (XMTINT%=1) or disabled (XMTINT%=0).

RCVINT% is an integer value specifying whether the receiver interrupt for the channel is enabled (RCVINT%=1) or disabled (RCVINT%=0).

BRKINT% is an integer value specifying whether the break condition interrupt for the channel is enabled (BRKINT%=1) or disabled (BRKINT%=0). When the break interrupt is enabled, the channel responds to a break interrupt by flushing the receive buffer and placing a CNTL-C character (03H) in the receive buffer.

CHAN% is an integer value specifying the number of the channel for which interrupt conditions are being defined (0-31).

Returned Value:

ARROR% contains the returned integer error code:

0 = no error
-1 = invalid channel selected

Example:

The following BASIC program segment could be written to enable transmitter and break condition interrupts and disable the receiver interrupt for channel 4.

```
40  XMTINT%=1
50  RCVINT%=0
60  BRKINT%=1
70  CHAN%=4
80  CALL PL7314INTEN(ARROR%,
XMTINT%,RCVINT%,BRKINT%, CHAN%)
```

PL7314RCV

Structure: **CALL PL7314RCV(STAT%,CHAN%)**

Description: This routine reads a received character from the buffer if present, otherwise it waits until a character is received. The routine is protected from never returning (if a character isn't received) by a timeout.

Input Parameters: **CHAN%** is an integer value specifying the number of the channel from which the character is to be received (0-31).

Returned Value: **STAT%** contains the returned status of the channel:

positive number =character received
-1 = invalid channel selected
-2 = timeout error
-3 = parity error
-4 = framing error
-5 = overrun error

Example: The following BASIC program segment could be written to wait for and then read a received character from channel 4.

```
40  CHAN%=4
50  CALL PL7314RCV(STAT%, CHAN%)
```

Structure: CALL PL7314RCVCHK(STAT%, CHAN%)

Description: This routine checks a serial channel to see if a character is available in the receive buffer.

Input Parameters: **CHAN%** is an integer value specifying the number of the channel to be checked for a received character (0-31).

Returned Value: **STAT%** contains the returned status of the channel:

0 = no character available
positive number = number of
  characters available in the
  receive buffer
-1 = invalid channel selected

Example: The following BASIC program segment could be written to check whether channel 4 has a received character available.

```
40  CHAN%=4
50  CALL PL7314RCVCHK(STAT%,
CHAN%)
```

Structure:

DIM BUFFER%(SIZE/2)
CALL PL7314SETBUF(ARROR%, SIZE%,BUFFER%(0),XMTRCV%, CHAN%)

Description:

This routine allocates a transmit or receive buffer that is a different size than the 8-character default buffer allocated by PL7314INIT. An n-character receive buffer requires 2
(n + 1) bytes and an n-character transmit buffer requires (n + 1) bytes.

Input Parameters:

SIZE% is an integer value specifying the size in bytes of the buffer to be allocated (1 - 32767).

BUFFER%(0) is the first element of the buffer being allocated.

XMTRCV% is an integer value specifying whether the buffer is to be a transmit buffer or a receive buffer (0 = transmit, 1 = receive).

CHAN% is a hexadecimal integer value specifying the number of the channel for which the buffer is being defined (0 - 31).

Returned Value:

ARROR% contains the returned integer error code:

0 = no error
-1 = invalid channel selected

Example:

The following BASIC program segment could be written to allocate a 49 character receive buffer for channel 4.

```
50 DIM BUFFER%(50)
60 SIZE%=100
70 XMTRCV%=1
80 CHAN%=4
90 CALL PL7314SETBUF(ARROR%,
SIZE%,BUFFER%(0),XMTRCV%,
CHAN%)
```

PL7314TEST

Structure: **CALL PL7314TEST(STAT%,CHAN%)**

Description: This routine performs a self test on one serial channel. The routine tests the control registers of the serial chip to see if they can be written to and read from. The data registers aren't written to in this test to avoid changing any outputs of the channel.

Input Parameters: **CHAN%** is an integer value specifying the number of the channel to be tested (0-31).

Returned Value: **STAT%** contains the returned status of the channel:

0 = no error
-1 = self test failed

Example: The following BASIC program segment could be written to test channel 4.

```
40  CHAN%=4
50  CALL PL7314TEST(STAT%, CHAN%)
```

Structure: **CALL PL7314XMT(STAT%,CHAN%, CHAR%)**

Description: This routine transmits a character by placing it in the transmit character buffer. If there is no room in the buffer, the routine waits until there is room or until a timeout indicates a system failure.

Input Parameters: **CHAN%** is an integer value specifying the number of the channel from which the character is to be transmitted (0-31).

CHAR% is the integer number whose value is the ASCII code for the character to be transmitted.

Returned Value: **STAT%** contains the returned status of the channel:

0 = no error
-1 = invalid channel selected
-2 = transmitter timeout error

Example: The following BASIC program segment could be written to transmit an ASCII "A" from channel 4.

```
40  CHAN%=4
50  CHAR%=ASC("A")
60  CALL PL7314XMT(STAT%,
CHAN%,CHAR%)
```

Structure:	**CALL PL7314XMTBRK(STAT%, CHAN%)**
Description:	This routine transmits a break condition immediately without waiting for the characters in the transmit buffer to be sent.
Input Parameters:	**CHAN%** is an integer value specifying the number of the channel from which the break is to be transmitted (0-31).
Returned Value:	**STAT%** contains the returned status of the channel: 0 = no error -1 = invalid channel selected
Example:	The following BASIC program segment could be written to transmit an immediate break condition from channel 4.

```
40  CHAN%=4
50  CALL PL7314XMTBRK(STAT%,
CHAN%)
```

Structure: **CALL PL7314XMTCHK(STAT%, CHAN%)**

Description: This routine checks to see if there is room in the transmit buffer for a character.

Input Parameters: **CHAN%** is an integer value specifying the number of the channel buffer to be checked (0-31).

Returned Value: **STAT%** contains the returned status of the channel:

0 = buffer has no room
positive number = number of empty locations in the buffer
-1 = invalid channel selected

Example: The following BASIC program segment could be written to check whether channel 4 has room in the transmit buffer for another character.

```
40  CHAN%=4
50  CALL PL7314XMTCHK(STAT%,
CHAN%)
```

PL7314XMTIMD

Structure:

CALL PL7314XMTIMD(STAT%, CHAN%,CHAR%)

Description:

This routine transmits a character immediately without placing it in the transmit character buffer.

Input Parameters:

CHAN% is an integer value specifying the number of the channel from which the character is to be transmitted (0-31).

CHAR% is the integer number whose value is the ASCII code for the character to be transmitted.

Returned Value:

STAT% contains the returned status of the channel:

 0 = no error
-1 = invalid channel selected

Example:

The following BASIC program segment could be written to transmit an ASCII "A" from channel 4 without using the transmit buffer.

```
40  CHAN%=4
50  CHAR%=ASC("A")
60  CALL PL7314XMTIMD(STAT%,
CHAN%,CHAR%)
```

The 7507 and 7508 General Purpose Interface Cards allow direct connection to one (7507) or two (7508) 24-I/O module mounting racks available from Opto-22, Gordos, Crydom, etc. Module mounting racks are also available in 8-module and 16-module versions. However, the software assumes that a 7507 has only one rack connected to it, and that a 7508 has no more than two racks. This requirement is independent of rack size.

The STD-LIB 7507 and 7508 software modules are as follows:

PL7508INIT:	Initializes the library software with the user-selected hardware configuration of the card.
PL7508RACKPWR:	Turns the rack power supplied to the user on or off.
PL7508READMOD:	Reads an input from a module or reads the status of an output to a module.
PL7508RESET:	Turns the module rack power off and sets all outputs to the modules low.
PL7508SETMOD:	Sets an output to a module to a 0 or a 1.
PL7508TEST:	This is a routine that simply returns a pass indication for self test, because the 7507 and 7508 cards can't be tested without changing the outputs.

PL7508INIT

Structure:

CALL PL7508INIT(ARROR%, BASEADR%,NORACK%)

Description:

This routine initializes the 7507 or 7508 library routines with the user-selected hardware configuration of the card if different from the as-shipped condition (BASEADR = &H50, NORACK = 2). This routine also resets the card to all module rack power switched off, all outputs low (off), and initializes storage for the previous state of the output bits to low.

Input Parameters:

BASEADR% is a hexadecimal integer value specifying the base address of the cards. This is the lowest port address of the contiguously mapped 7507 and 7508 cards. The library routines use a 10-bit address even if the card only recognizes 8 bits of address.

NORACK% is an integer value specifying the number of racks controlled by contiguously mapped cards (1 - 44).

Returned Value:

ARROR% contains the returned returned integer error code:

0 = no error

Example:

The following BASIC program segment could be written to initialize the library for two 7508 cards, the first card mapped at 160-167 and the second mapped at 168-16F. These cards can control four 24-position racks.

```
40  BASEADR%=&H160
50  NORACK%=4
60  CALL PL7508INIT(ARROR%,
BASEADR%,NORACK%)
```

PL7508RACKPWR

Structure:	**CALL PL7508RACKPWR(ARROR%, RACK%,ONOFF%)**
Description:	This routine turns rack power on or off.
Input Parameters:	**RACK%** is an integer value specifying the rack power to be enabled or disabled (0 - 43). **ONOFF%** is an integer value specifying whether rack power is to be turned on (ONOFF% = 1) or off (ONOFF% = 0).
Returned Value:	ARROR% contains the returned integer error code: 0 = no error -1 = invalid rack selected
Example:	The following BASIC program segment could be written to turn on power to rack 7.

```
40  RACK%=7
50  ONOFF%=1
60  CALL PL7508RACKPWR(
ARROR%,RACK%,ONOFF%)
```

PL7508READMOD

Structure: **CALL PL7508READMOD(STAT%, RACK%,MODULE%)**

Description: This routine reads an input module.

Input Parameters: **RACK%** is an integer value specifying the rack containing the module to be read (0 - 43).

MODULE% is an integer value specifying the input module to be read (0 - 23).

Returned Value: **STAT%** contains the results of the read:

- 0 = bit clear
- 1 = bit set
- -1 = invalid module or rack selected

Example: The following BASIC program segment could be written to read input module 22 of the middle rack in a three rack system.

```
40  RACK%=1
50  MODULE%=22
60  CALL PL7508READMOD(STAT%,
RACK%,MODULE%)
```

Structure: **CALL PL7508RESET(ARROR%)**

Description: This routine software resets all contiguously mapped 7507 or 7508 cards. All module outputs are set low (off), and all rack power is turned off.

Input Parameters: None

Returned Value: **ARROR%** contains the returned integer error code:

0 = no error

Example: The following BASIC program segment could be written to reset all contiguously mapped 7507 or 7508 cards.

```
40  CALL PL7508RESET(ARROR%)
```

PL7508SETMOD

Structure:	**CALL PL7508SETMOD(ARROR%, RACK%,MODULE%,ONOFF%)**
Description:	This routine writes a 1 or a 0 to a module output.
Input Parameters:	**RACK%** is an integer value specifying the rack containing the module to be written to (0 - 43). **MODULE%** is an integer value specifying the output module to be written (0 - 23). **ONOFF%** is an integer value specifying the value to be written to the module (0 or 1).
Returned Value:	**ARROR%** contains the returned integer error code: 0 = no error -1 = invalid module or rack selected -2 = readback error (always returned if rack power is off)
Example:	The following BASIC program segment could be written to set output 22 of the middle rack in a three-rack system to a 1 (turn on module 22 of rack 1).

```
40  RACK%=1
50  MODULE%=22
60  ONOFF%=1
70  CALL PL7508SETMOD(ARROR%,
RACK%,MODULE%,ONOFF%)
```

Structure:	**CALL PL7508TEST(ARROR%)**
Description:	This routine simply returns a pass indication for self-test, because the 7507 and 7508 cards can't be tested without changing the outputs. This routine is included because self-test routines are required for all cards in the library.
Input Parameters:	None
Returned Value:	**ARROR%** contains the returned integer error code: 0 = no error
Example:	The following BASIC program segment could be written to call a self-test routine for 7507 or 7508 cards.

```
40  CALL PL7508TEST(ARROR%)
```

PRO-LOG
PL7700
Memory Diagnostic Routines
BASIC library

The Memory Diagnostic routines are designed to test all the memory cards in the STD-DOS system, this gives users a high level of confidence that their system is fully functional. The memory diagnostic routines are:

PL7700CHECKSUM:	Computes and reports the checksum of a specified area of the DOS system.
PL7700MEMTEST:	Tests all system RAM and reports to the user the amount of RAM (in kbytes) that was successfully tested.

PL7700CHECKSUM

Structure:

CALL **PL7700CHECKSUM**(CHKSUM, SEGNO%,NOPARGRF%)

Description:

This routine computes the two's complement checksum of a specified area of the DOS system. The checksum is reported as a single precision floating point number.

Input Parameters:

SEGNO% is an integer value specifying the start segment of the area to be checksummed (offset = 0000).

NOPARGRF% is an integer value specifying the number of 16-byte paragraphs to be added to the checksum.

Returned Value:

CHKSUM contains the checksum of the specified area of memory.

Example:

The following BASIC program segment could be written to checksum the memory from C000:0000 to C000:FFFF.

```
40 SEGNO%=&HC000
50 NOPARGRF%=&H1000
60 CALL PL7700CHECKSUM(CHKSUM,
SEGNO%,NOPARGRF%)
```

Structure:	CALL **PL7700MEMTEST(MEMSIZE%)**
Description:	This routine tests all contiguous RAM in the STD-DOS system. The test starts at segment 0000 and continues until the top of RAM or an error is detected. For each memory location, the memory test writes then verifies a 00 hex then an FF hex; once tested, each memory location is returned to its original value.
Input Parameters:	None
Returned Value:	**MEMSIZE%** contains the amount of RAM tested successfully (in kbytes).
Example:	The following BASIC program segment could be written to test all contiguous RAM beginning at 0000:0000.

```
CALL PL7700MEMTEST(MEMSIZE%)
```

PRO-LOG
7863
CPU Card
BASIC library

The 7863 CPU card is equipped with a real-time clock device. The STD-LIB 7863 software provides the user with the ability to read or write to the clock, to wait for a specified period of time, and to set future time marks which later may be checked to see if the time has expired. All 7863 library routines except SETMARK and SETALARM access the clock directly.

The STD-LIB 7863 software modules are as follows:

PL7863CHKMARK:	Checks for expiration of a previously set time mark.
PL7863GETDATE:	Gets the current date.
PL7863GETTIME:	Gets the current time.
PL7863GETWATCH:	Gets the date, time, and weekday in one call.
PL7863INIT:	Sets the current date and time on the clock. Resets all time marks.
PL7863SETALARM:	Sets an internal alarm mark to enable software to wait until a specific day and time.
PL7863SETMARK:	Sets an internal future time mark for a specified time interval from the current day and time.
PL7863TEST:	Performs a self-test on the clock. The routine gets the time, waits a while, then gets the time again. It passes self-test if the time has increased.
PL7863WAIT:	Waits for a specified period of time.
PL7863WEEKDAY:	Gets the current day of the week.

The clock maintains the current year in the form 0-99. For convenience, the year as specified (and returned) by all routines is in the form 1980-2079. In addition, the clock maintains the current hour in 24-hour mode (0-23). This is reported to the caller in this fashion.

This clock supports time measurement with resolution of a hundredth of a second. For the PL7863WAIT routine, time intervals shorter than the minimum measurement will be measured using software program loops.

The PL7863SETALARM and PL7863SETMARK (set interval) routines share the same internal structure. If the user attempts to set an interval mark using the same mark number as a previously set, unexpired alarm mark, the old mark will be overwritten by the new one. Marks may be reset at any time via calls to PL7863SETMARK or PL7863SETALARM. It is the user's responsibility not to overwrite an existing mark before it has expired.

Structure:

CALL PL7863CHKMARK(ARROR%, MARKNUM%)

Description:

This routine checks for expiration of a previously set time mark. It verifies that the mark is set, then compares it with the current time. If the time is greater than the mark, the mark will be cleared.

Input Parameters:

MARKNUM% is an integer value specifying the time mark to be checked (0-9).

Returned Value:

ARROR% contains the returned integer error code:

0 = interval has not expired
1 = interval has expired
-1 = invalid mark number
-2 = mark not set

Example:

The following BASIC program segment could be written to check mark number 2 to see if it has expired.

```
40 MARKNUM%=2
50 CALL PL7863CHKMARK (ARROR%,
MARKNUM%)
```

PL7863GETDATE

Structure:	**CALL PL7863GETDATE (YEAR%, MONTH%, DAY%)**
Description:	This routine gets the current date.
Input Parameters:	None
Returned Value:	**YEAR%** contains the current year (1980-2079). **MONTH%** contains the current month (1-12). **DAY%** contains the current day (1-31).
Example:	The following BASIC program segment could be written to get the current date.

```
40 CALL PL7863GETDATE(YEAR%,
MONTH%, DAY%)
```

Structure:	**CALL PL7863GETTIME (HOURS%, MINUTES%, HUNDREDTHS%)**
Description:	This routine gets the current time.
Input Parameters:	None
Returned Value:	**HOURS%** contains the current hours (0-23). **MINUTES%** contains the current minutes (0-59). **HUNDREDTHS%** contains the current hundredths of a second (0-5999).
Example:	The following BASIC program segment gets the current time: `40 CALL PL7863GETTIME(HOURS%, MINUTES%, HUNDREDTHS%)`

PL7863GETWATCH

Structure:

```
DIM WATCHARRAY%(8)
CALL PL7863GETWATCH(ARROR%,
WATCHARRAY%(0))
```

Description:

This routine is used to obtain the date, time, and weekday in one call. Since the data cannot easily be passed, the caller should pass the address of an eight-word array.

Input Parameters:

None

Returned Value:

ARROR% contains the returned integer error code:

0 = no error
-1 = pointer is illegal (pointer equals 0)

WATCHARRAY% contains the following information in order, if no error:

WATCHARRAY(0) = year (1980-2079)
WATCHARRAY(1) = month (1-12)
WATCHARRAY(2) = day (1-31)
WATCHARRAY(3) = hours (0-23)
WATCHARRAY(4) = minutes (0-59)
WATCHARRAY(5) = seconds (0-59)
WATCHARRAY(6) = hundredths of a second (0-99)
WATCHARRAY(7) = weekday (1=Sunday, 2=Monday, etc.)

Example:

The following BASIC program segment could be written to read all the time data.

```
30 DIM WATCHARRAY%(8)
40 CALL PL7863GETWATCH (ARROR%,
WATCHARRAY%(0))
```

Structure:	**CALL PL7863INIT (ARROR%, YEAR%, MONTH%, DAY%, HOURS%, MINUTES%, SECONDS%, HUNDREDTHS%)**
Description:	This routine sets the current date and time on the clock and clears any time marks that have been set. It also checks the clock with a call to the PL7863TEST routine. Note that this routine will change the time and date known to the STD-DOS operating system.
Input Parameters:	**YEAR%** is an integer value specifying the current year (1980-2079). **MONTH%** is an integer value specifying the current month (1-12). **DAY%** is an integer value specifying the current day (1-31). **HOURS%** is an integer value specifying the current hours (0-23). **MINUTES%** is an integer value specifying the current minutes (0-59). **SECONDS%** is an integer value specifying the current seconds (0-59). **HUNDREDTHS%** is an integer value specifying the current hundredths of a second (0-99).

Returned Value: ARROR% contains the returned integer error code:

 0 = no error
-1 = clock is not operational
-2 = invalid date
-3 = invalid time

Example: The following BASIC program segment could be written to set the current time to 07:00:00, September 1, 1986. The 7863 library software is also initialized.

```
40 YEAR%=1986
50 MONTH%=9
60 DAY%=1
70 HOURS%=7
80 MINUTES%=0
90 SECONDS%=0
100 HUNDREDTHS%=0
110 CALL PL7863INIT(ARROR%, YEAR%,
MONTH%, DAY%, HOURS%, MINUTES%,
SECONDS%, HUNDREDTHS%)
```

Structure: **CALL PL7863SETALARM(ARROR%, MARKNUM%, WEEKDAY%, HOURS%, MINUTES%, SECONDS%)**

Description: This routine sets an alarm mark for the specified time. PL7863CHKMARK can be used to determine if the alarm has occurred. Up to ten time marks may be set at one time. These ten time marks are set by both this routine and the PL7863SETMARK routine. If the specified time is earlier than the current time, the alarm will be set for the next coming week.

Input Parameters: **MARKNUM%** is an integer value specifying the time mark to be set (0-9).

WEEKDAY% is an integer value indicating the day of the week (1-7) of the alarm time.

HOURS% is an integer value indicating the hour (0-23) of the alarm time.

MINUTES% is an integer value indicating the minutes (0-59) of the alarm time.

SECONDS% is an integer value indicating the seconds (0-59) of the alarm time.

Returned Value: **ARROR%** contains the returned integer error code:

0 = no error
-1 = invalid mark number
-2 = invalid interval

Example: The following BASIC program segment could be written to set mark number 2 for Monday, 7 a.m.

```
40 MARKNUM%=2
50 WEEKDAY%=2
60 HOURS%=7
70 MINUTES%=0
80 SECONDS%=0
90 CALL PL7863SETALARM(ARROR%,
MARKNUM%, WEEKDAY%, HOURS%,
MINUTES%, SECONDS%)
```

Structure:	**CALL PL7863SETMARK(ARROR%, MARKNUM%, DAYS%, HOURS%, MINUTES%, SECONDS%)**
Description:	This routine sets a time mark, measured from the current time, whose expiration can be checked via calls to PL7863CHKMARK. Up to ten time marks may be set at a time. Note that the same ten time marks may be set via calls to PL7863SETALARM. The maximum time for a mark interval is one week.
Input Parameters:	**MARKNUM%** is an integer value specifying the time mark to be set (0-9). **DAYS%** is an integer value indicating the number of days (0-6) in the time interval. **HOURS%** is an integer value specifying the number of hours (0-23) in the time interval. **MINUTES%** is an integer value indicating the number of minutes (0-59) in the time interval. **SECONDS%** is an integer value indicating the number of seconds (0-59) in the time interval.
Returned Value:	**ARROR%** contains the returned integer error code: 0 = no error -1 = invalid mark number -2 = invalid time interval

Example:

The following BASIC program segment could be written to set mark number 3 for one minute from now.

```
40 MARKNUM%=3
50 DAYS%=0
60 HOURS%=0
70 MINUTES%=1
80 SECONDS%=0
90 CALL PL7863SETMARK(ARROR%,
MARKNUM%, DAYS%, HOURS%,
MINUTES%, SECONDS%)
```

Structure:	**CALL PL7863TEST(ARROR%)**
Description:	This routine determines if the clock is operational. It will get the time, wait a while (known to be more than a hundredth of a second), and then get the time again. The clock is presumed to be operating correctly if the time has increased.
Input Parameters:	None
Returned Value:	ARROR% contains the returned integer error code: 0 = no error -1 = clock is not operational
Example:	The following BASIC program segment tests the clock. 40 CALL PL7863TEST (ARROR%)

PL7863WAIT

Structure:

CALL PL7863WAIT (ARROR%, SECONDS%, MILLI%, MICRO%)

Description:

This routine can be used to wait for a specified period of time. The maximum waiting time interval is 60 seconds.

Input Parameters:

SECONDS% is an integer value indicating the number of seconds (0-59) in the wait interval.

MILLI% is an integer value indicating the number of milliseconds (0-999) in the wait interval.

MICRO% is an integer value indicating the number of microseconds (0-999) in the wait interval.

Returned Value:

ARROR% contains the returned integer error code:

 0 = no error
-1 = invalid interval

Example:

The following BASIC program segment could be written to wait for 2.5 seconds.

```
40 SECONDS%=2
50 MILLI%=500
60 MICRO%=0
70 CALL PL7863WAIT(ARROR%,
SECONDS%, MILLI%, MICRO%)
```

Structure:	**CALL PL7863WEEKDAY (WKDAY%)**
Description:	This routine gets the current day of the week.
Input Parameters:	None
Returned Value:	**WKDAY%** contains the current day of the week (1-7), 1 = Sunday, 2 = Monday, etc.
Example:	The following BASIC program segment gets the current day of the week.

```
40 CALL PL7863WEEKDAY (WKDAY%)
```

PRO-LOG
7888
CPU Card
BASIC library

The 7888 CPU card is equipped with a real-time clock device. The STD-LIB 7888 software provides the user with the ability to read or write to the clock, to wait for a specified period of time, and to set future time marks which later may be checked to see if the time has expired. All 7888 library routines except SETMARK and SETALARM access the clock directly.

The STD-LIB 7888 software modules are as follows:

PL7888CHKMARK:	Checks for expiration of a previously set time mark.
PL7888GETDATE:	Gets the current date.
PL7888GETTIME:	Gets the current time.
PL7888GETWATCH:	Gets the date, time, and weekday in one call.
PL7888INIT:	Sets the current date and time on the clock, then resets all time marks.
PL7888SETALARM:	Sets an internal alarm mark for a specified day and time.
PL7888SETMARK:	Sets an internal future time mark for a specified time interval from the current day and time.
PL7888TEST:	Performs a self-test on the clock by reading the status bit on the clock.
PL7888WAIT:	Waits for a specified period of time.
PL7888WEEKDAY:	Gets the current day of the week.

The clock maintains the current year in the form 0-99. For convenience, the year as specified (and returned) by all routines is in the form 1980-2079. In addition, the clock maintains the current hour in 24-hour mode (0-23). This is reported to the caller in this fashion. This clock supports time measurement with resolution of one second. For the PL7888WAIT routine, time intervals shorter than the minimum measurement will be measured using software program loops.

The PL7888SETALARM and PL7888SETMARK (set interval) routines share the same internal structure. If the user attempts to set an interval mark using the same mark number as a previously set, unexpired alarm mark, the old mark will be overwritten by the new one. Marks may be reset at any time via calls to PL7888SETMARK or PL7888SETALARM. It is the user's responsibility not to overwrite an existing mark before it has expired.

The watch chip on the 7888 CPU board will only resolve times down to one second. However, a few subroutine linkages feature times of hundredths of a second. This has been done to make the 7863 CPU board support and the 7888 CPU board support as similar as possible.

Structure:

CALL PL7888CHKMARK (ARROR%, MARKNUM%)

Description:

This routine checks for expiration of a previously set time mark. It verifies that the mark is set, then compares it with the current time. If the time is greater than the mark, the mark will be cleared.

Input Parameters:

MARKNUM% is an integer value specifying the time mark to be checked (0-9).

Returned Value:

ARROR% contains the returned integer error code:

0 = interval has not expired
1 = interval has expired
-1 = invalid mark number
-2 = mark not set

Example:

The following BASIC program segment could be written to check mark number 2 to see if it has expired.

```
40 MARKNUM%=2
50 CALL PL7888CHKMARK (ARROR%,
MARKNUM%)
```

PL7888GETDATE

Structure:	**CALL PL7888GETDATE (YEAR%, MONTH%, DAY%)**
Description:	This routine gets the current date.
Input Parameters:	None
Returned Value:	**YEAR%** contains the current year (1980-2079). **MONTH%** contains the current month (1-12). **DAY%** contains the current day (1-31).
Example:	The following BASIC program segment could be written to get the current date.

```
40 CALL PL7888GETDATE(YEAR%,
MONTH%, DAY%)
```

PL7888GETTIME

Structure:	**CALL PL7888GETTIME (HOURS%, MINUTES%, HUNDREDTHS%)**
Description:	This routine gets the current time. Due to linkage limitations, the returned time will have a "hundredths of a second" field rather than a "seconds" field.
Input Parameters:	None
Returned Value:	**HOURS%** contains the current hours (0-23). **MINUTES%** contains the current minutes (0-59). **HUNDREDTHS%** contains the current hundredths of a second (0-5999).
Example:	The following BASIC program segment gets the current time: 40 CALL PL7888GETTIME(HOURS%, MINUTES%, HUNDREDTHS%)

PL7888GETWATCH

Structure:

```
DIM WATCHARRAY%(8)
CALL PL7888GETWATCH(ARROR%,
WATCHARRAY%(0))
```

Description:

This routine is used to obtain the date, time, and weekday in one call. Since the data cannot easily be passed, the caller should pass the address of an eight-word array.

Input Parameters:

None

Returned Value:

ARROR% contains the returned integer error code:

 0 = no error
-1 = pointer is illegal (pointer equals 0)

WATCHARRAY% contains the following information in order, if no error:

WATCHARRAY(0) = year (1980-2079)
WATCHARRAY(1) = month (1-12)
WATCHARRAY(2) = day (1-31)
WATCHARRAY(3) = hours (0-23)
WATCHARRAY(4) = minutes (0-59)
WATCHARRAY(5) = seconds (0-59)
WATCHARRAY(6) = hundredths of a second (0-99)
WATCHARRAY(7) = weekday (1=Sunday, 2=Monday, etc.)

Example:

The following BASIC program segment could be written to read all the time data.

```
30 DIM WATCHARRAY%(8)
40 CALL PL7888GETWATCH (ARROR%,
WATCHARRAY%(0))
```

Structure:	**CALL PL7888INIT (ARROR%, YEAR%, MONTH%, DAY%, HOURS%, MINUTES%, SECONDS%, HUNDREDTHS%)**
Description:	This routine sets the current date and time on the clock and clears any time marks that have been set. It also checks the clock with a call to the PL7888TEST routine. Note that this routine will change the time and date known to the STD-DOS system when the STD-DOS system is subsequently reset.
Input Parameters:	**YEAR%** is an integer value specifying the current year (1980-2079). **MONTH%** is an integer value specifying the current month (1-12). **DAY%** is an integer value specifying the current day (1-31). **HOURS%** is an integer value specifying the current hours (0-23). **MINUTES%** is an integer value specifying the current minutes (0-59). **SECONDS%** is an integer value specifying the current seconds (0-59). **HUNDREDTHS%** is an integer value specifying the current hundredths of a second (0-99).

Returned Value: **ARROR%** contains the returned integer error code:

- 0 = no error
- -1 = clock is not operational
- -2 = invalid date
- -3 = invalid time

Example: The following BASIC program segment could be written to set the current time to 07:00:00, September 1, 1986. The 7888 library software is also initialized.

```
40 YEAR%=1986
50 MONTH%=9
60 DAY%=1
70 HOURS%=7
80 MINUTES%=0
90 SECONDS%=0
100 HUNDREDTHS%=0
110 CALL PL7888INIT(ARROR%, YEAR%,
MONTH%, DAY%, HOURS%, MINUTES%,
SECONDS%, HUNDREDTHS%)
```

Structure:	**CALL PL7888SETALARM(ARROR%, MARKNUM%, WEEKDAY%, HOURS%, MINUTES%, SECONDS%)**
Description:	This routine sets an alarm mark for the specified time. PL7888CHKMARK can be used to determine if the alarm has occurred. Up to ten time marks may be set at one time. These ten time marks are set by both this routine and the PL7888SETMARK routine. If the specified time is earlier than the current time, the alarm will be set for the next coming week.
Input Parameters:	**MARKNUM%** is an integer value specifying the time mark to be set (0-9). **WEEKDAY%** is an integer value indicating the day of the week (1-7) of the alarm time. **HOURS%** is an integer value indicating the hour (0-23) of the alarm time. **MINUTES%** is an integer value indicating the minutes (0-59) of the alarm time. **SECONDS%** is an integer value indicating the seconds (0-59) of the alarm time.
Returned Value:	**ARROR%** contains the returned integer error code: 0 = no error -1 = invalid mark number -2 = invalid interval

Example: The following BASIC program segment could be written to set mark number 2 for Monday, 7 a.m.

```
40 MARKNUM%=2
50 WEEKDAY%=2
60 HOURS%=7
70 MINUTES%=0
80 SECONDS%=0
90 CALL PL7888SETALARM(ARROR%,
MARKNUM%, WEEKDAY%, HOURS%,
MINUTES%, SECONDS%)
```

Structure:	**CALL PL7888SETMARK(ARROR%, MARKNUM%, DAYS%, HOURS%, MINUTES%, SECONDS%)**
Description:	This routine sets a time mark, measured from the current time, whose expiration can be checked via calls to PL7888CHKMARK. Up to ten time marks may be set at a time. Note that the same ten time marks may be set via calls to PL7888SETALARM. The maximum time for a mark interval is one week.
Input Parameters:	**MARKNUM%** is an integer value specifying the time mark to be set (0-9). **DAYS%** is an integer value indicating the number of days (0-6) in the time interval. **HOURS%** is an integer value specifying the number of hours (0-23) in the time interval. **MINUTES%** is an integer value indicating the number of minutes (0-59) in the time interval. **SECONDS%** is an integer value indicating the number of seconds (0-59) in the time interval.
Returned Value:	**ARROR%** contains the returned integer error code: 0 = no error -1 = invalid mark number -2 = invalid time interval

PL7888SETMARK

Example:

The following BASIC program segment could be written to set mark number 3 for one minute from now.

```
40 MARKNUM%=3
50 DAYS%=0
60 HOURS%=0
70 MINUTES%=1
80 SECONDS%=0
90 CALL PL7888SETMARK(ARROR%,
MARKNUM%, DAYS%, HOURS%,
MINUTES%, SECONDS%)
```

PL7888TEST

Structure:	**CALL PL7888TEST(ARROR%)**
Description:	This routine determines if the clock is operational by reading the clock status.
Input Parameters:	None
Returned Value:	ARROR% contains the returned integer error code: 0 = no error -1 = clock is not operational
Example:	The following BASIC program segment tests the clock.

```
40 CALL PL7888TEST (ARROR%)
```

PL7888WAIT

Structure:

CALL PL7888WAIT (ARROR%, SECONDS%, MILLI%, MICRO%)

Description:

This routine can be used to wait for a specified period of time. The maximum waiting time interval is 60 seconds.

Input Parameters:

SECONDS% is an integer value indicating the number of seconds (0-59) in the wait interval.

MILLI% is an integer value indicating the number of milliseconds (0-999) in the wait interval.

MICRO% is an integer value indicating the number of microseconds (0-999) in the wait interval.

Returned Value:

ARROR% contains the returned integer error code:

 0 = no error
-1 = invalid interval

Example:

The following BASIC program segment could be written to wait for 2.5 seconds.

```
40 SECONDS%=2
50 MILLI%=500
60 MICRO%=0
70 CALL PL7888WAIT(ARROR%,
SECONDS%, MILLI%, MICRO%)
```

Structure:	**CALL PL7888WEEKDAY (WKDAY%)**
Description:	This routine gets the current day of the week.
Input Parameters:	None
Returned Value:	**WKDAY%** contains the current day of the week (1-7), 1 = Sunday, 2 = Monday, etc.
Example:	The following BASIC program segment gets the current day of the week.

```
40 CALL PL7888WEEKDAY (WKDAY%)
```

Section 6

ASSEMBLY LANGUAGE LIBRARY

CONTENTS	**PAGE**
Introduction	**6-1**
Analog Devices RTI-1225 Analog Input/Output Card	**6-1226-1**
AD1225_AIN	6-1226-4
AD1225_AOUT	6-1226-6
AD1225_INIT	6-1226-8
AD1225_RESET	6-1226-13
AD1225_TEST	6-1226-14
Analog Devices RTI-1226 Analog Input Card	**6-1226-1**
AD1226_AIN	6-1226-4
AD1226_INIT	6-1226-5
AD1226_RESET	6-1226-8
AD1226_TEST	6-1226-9
Analog Devices RTI-1260 Analog Input Card	**6-1260-1**
AD1260_AIN	6-1260-4
AD1260_INIT	6-1260-5
AD1260_RESET	6-1260-8
AD1260_TEST	6-1260-9
Analog Devices RTI-1262 Analog Output Card	**6-1262-1**
AD1262_AMPS	6-1262-4
AD1262_AOUT	6-1262-6
AD1262_INIT	6-1262-7
AD1262_RESET	6-1262-10

Pro-Log 75XX, 76XX, and 79XX Input/Output Cards 6-7000-1

PL7000_INIT 6-7000-3
PL7000_RDBIT 6-7000-5
PL7000_RDPORT 6-7000-6
PL7000_RESETPORT 6-7000-7
PL7000_WRTBIT 6-7000-8
PL7000_WRTBITRD 6-7000-10
PL7000_WRTPORT 6-7000-12
PL7000_WRTPORTRD 6-7000-13
7501 DC Driver Card 6-7000-14
7502 SPST Relay Output Card 6-7000-14
7503 Optoisolated Input Card 6-7000-15
7504 Triac Output Card 6-7000-15
7506 Optoisolated Input Card 6-7000-16
7601 TTL Input/Output Port Card 6-7000-16
7602A TL Output Port Card 6-7000-17
7603A TTL Input Port Card 6-7000-17
7604A TTL Input/Output Card 6-7000-18
7605 Programmable TTL I/O Card 6-7000-19
7614 66-Line TTL I/O Card 6-7000-20
7904 Decoded I/O Utility Card 6-7000-21
7914 Decoded I/O Utility Card 6-7000-22

Pro-Log 7303 Keyboard Display Card 6-7303-1

PL7303_CHRSHFT 6-7303-3
PL7303_CHRWR 6-7303-4
PL7303_CURSOR 6-7303-5
PL7303_GETKEY 6-7303-7
PL7303_GETROCKR 6-7303-8
PL7303_INIT 6-7303-9
PL7303_KEYFETCH 6-7303-11
PL7303_KEYSTAT 6-7303-12

PL7303_MESSAGE 6-7303-13
PL7303_RCKRSTAT 6-7303-15
PL7303_RESET 6-7303-16
PL7303_SETLEDS 6-7303-17
PL7303_TEST 6-7303-18
PL7303_WAIT 6-7303-19
PL7303_WORD 6-7303-20

Pro-Log 7312/7314A Communications Card 6-7316-1

PL7314_BUFFLUSH 6-7316-4
PL7314_CHANRST 6-7316-6
PL7314_INIT 6-7316-8
PL7314_INTEN 6-7316-10
PL7314_RCV 6-7316-12
PL7314_RCVCHK 6-7316-13
PL7314_SETBUF 6-7316-14
PL7314_TEST 6-7316-16
PL7314_XMT 6-7316-17
PL7314_XMTBRK 6-7316-18
PL7314_XMTCHK 6-7316-19
PL7314_XMTIMD 6-7316-20

Pro-Log 7507/7508 TTL Input/Output Card 6-7508-1

PL7508_INIT 6-7508-2
PL7508_RACKPWR 6-7508-4
PL7508_READMOD 6-7508-5
PL7508_RESET 6-7508-6
PL7508_SETMOD 6-7508-7
PL7508_TEST 6-7508-9

Pro-Log 7700 Memory Diagnostic Routines 6-7700-1

PL7700_CHECKSUM 6-7700-2
PL7700_MEMTEST 6-7700-3

Pro-Log 7863 CPU Card 6-7863-1

PL7863_CHKMARK 6-7863-3
PL7863_GETDATE 6-7863-4
PL7863_GETTIME 6-7863-6
PL7863_GETWATCH 6-7863-8
PL7863_INIT 6-7863-10
PL7863_SETALARM 6-7863-12
PL7863_SETMARK 6-7863-14
PL7863_TEST 6-7863-16
PL7863_WAIT 6-7863-17
PL7863_WEEKDAY 6-7863-18

Pro-Log 7888 CPU Card 6-7888-1

PL7888_CHKMARK 6-7888-3
PL7888_GETDATE 6-7888-4
PL7888_GETTIME 6-7888-6
PL7888_GETWATCH 6-7888-8
PL7888_INIT 6-7888-10
PL7888_SETALARM 6-7888-12
PL7888_SETMARK 6-7888-14
PL7888_TEST 6-7888-16
PL7888_WAIT 6-7888-17
PL7888_WEEKDAY 6-7888-18

ASSEMBLY LANGUAGE LIBRARY

INTRODUCTION

This section contains the STD-LIB routines for application programs written in assembly language.

Your assembly language application program should be linked with the STD-LIB assembly language version, **STDLIBAL.LIB**. Refer to Section 2, Operation and Programming, for a description of how to link the assembly language version of STD-LIB with your application program.

Analog Devices RTI-1225

Analog Input/Output Card

Assembly library

The Analog Devices RTI-1225 Analog Input/Output Card is a general-purpose, low-cost analog I/O card. The RTI-1225 card uses an analog multiplexer to select between analog input channels that are connected to a 10-bit A/D converter. The input channels can be configured to be 16 single-ended or 8 differential inputs. There are two 8-bit D/A converters providing two output channels.

The STD-LIB RTI-1225 software modules are as follows:

AD1225_AIN:	Reads an input voltage and converts the A/D converter binary value to a reading rounded to the nearest millivolt (based on the default hardware configuration or the configuration set by AD1225_INIT).
AD1225_AOUT:	Transforms the output voltage requested in the call from millivolts to the nearest D/A code and writes an output voltage.
AD1225_INIT:	Initializes the library software with the user-selected hardware configuration of the card. If this routine has been invoked properly, a call is made to the AD1225_RESET routine.
AD1225_RESET:	Performs a software reset of the card. The multiplexer is set to select input channel 0 and wait for any pending conversions to complete or a conversion busy bit timeout. The D/A input registers are set to output a voltage of 0V.

AD1225_TEST:	Performs a self-test on one card. The routine reads a voltage from the card and passes self-test if there is no timeout error.

NOTE: The RTI-1225 as-shipped mapping is at hexadecimal base address FFF0 in the memory map. To be compatible with the library, the card needs to be port mapped with an 8-bit address. The default 8-bit base address for the library is hexadecimal B0 which results in the card using ports BB through BF.

The Analog Devices convention is to define the base address of a card as an address with the last four bits being zero. The library has followed this convention. This means that an RTI-1225 card that has been remapped to the library default would be defined as a base address of 1B0 in the initialize command, but the physical ports of the card are actually located at 1BB through 1BF. When referring to contiguous mapping for RTI-1225 cards, the cards are mapped as though they occupied 16 ports per card, even though they only physically occupy five ports per card.

To change an RTI-1225 card from its as-shipped mapping to the library default mapping, make the following changes to the jumpers:

remove 59-60,62-63,51-52
install 61-63,58-59,53-55,15-16

For technical questions concerning STD-LIB software for the Analog Devices RTI-1225, contact Pro-Log customer service. For technical questions concerning the operation of the Analog Devices RTI-1225 hardware, contact the appropriate Analog Devices customer service as follows:

STATE	PHONE NUMBER
Alaska	
Arizona	(714)-641-9391
California (southern)	Direct Sales Office
Colorado	
California (northern)	(408)-947-0633
Utah	Direct Sales Office
Delaware	(215)-643-7790
New Jersey	Direct Sales Office
Pennsylvania	
Maine	
Massachusetts	(617)-329-4700
New Hampshire	Factory
Rhode Island	
Illinois	
Indiana	
Iowa	
Kansas	
Kentucky	(312)-653-5000
Missouri	Direct Sales Office
North Dakota	
South Dakota	
Wisconsin	
All Other States	(617)-329-4700
	Factory

AD1225_AIN

Structure:

```
EXTRN      AD1225_AIN:FAR
MOV        DX,CARDNO
MOV        CX,CHANNEL
CALL       AD1225_AIN
```

Description:

This routine reads an input voltage from the selected channel. The binary reading of the A/D converter is translated to a binary reading in millivolts based on the user-selected input configuration.

Input Parameters:

The **DX** register specifies the card from which the reading is to be taken (0 - 21).

The **CX** register specifies the input channel to be read (0 - 15).

Returned Value:

The **AX** register contains the analog voltage read in millivolts (-10000 to + 9990) or contains an error code:

-32768 = channel number out of range
-32767 = board number out of range
-32766 = timeout error during conversion

Example:

The following Assembly language program segment could be written to read an analog voltage from input channel 2 of the fourth card in the system.

```
MOV   DX,3
MOV   CX,2
CALL  AD1225_AIN
```

AD1225_AOUT

Structure:

```
EXTRN        AD1225_AOUT:FAR
MOV     DX,CARDNO
MOV     CX,CHANNEL
MOV     DI,VOLTAGE
CALL    AD1225_AOUT
```

Description:

This routine outputs a voltage to the selected channel. The selected voltage is input in millivolts and converted to a binary program value for the D/A converter based on the user-selected output configuration for the channel.

Input Parameters:

The **DX** register specifies the card from which the voltage is to be output (0 - 21).

The **CX** register specifies the channel from which the voltage is to be output (0 or 1).

The **BX** register contains the analog voltage to be output in millivolts (-10000 to + 9961).

Returned Value:

The **AX** register contains the returned integer error code:

0 = no error
-1 = channel number out of range
-2 = board number out of range
-3 = voltage parameter too large
(clipped to maximum value)
-4 = voltage parameter too small
(clipped to minimum value)

Example:

The following Assembly language program segment could be written to output an analog voltage of 2V from output channel 1 of the fourth card in the system.

```
MOV   DX,3
MOV   CX,1
MOV   DI,2000
CALL  AD1225_AOUT
```

Structure:

```
EXTRN         AD1225_INIT:FAR
MOV       CH,BASENO
MOV       BX,BASEADR
MOV       CL,NOCARDS
MOV       DH,INCHAN
MOV       DL,ADCONF
MOV       AH,DA0CONF
MOV       AL,DA1CONF
CALL      AD1225_INIT
```

Description:

This routine initializes the RTI-1225 library routines with the user-selected hardware configuration of the card if different from the default condition. The card default initialization is for one RTI-1225 card in a system with a base address of 1B0, configured for 16 single-ended inputs of ±10V two's complement and both outputs ±10V two's complement. There may be up to 22 of these cards in a system. When using more than one card, all cards with the same configuration are mapped contiguously and are initialized with the same routine call. If there are no parameter errors, the card is software reset with a call to the AD1225_RESET routine.

Input Parameters:

The **CH** register specifies the base number of the group of contiguous cards configured by this command (0 - 21).

The **BX** register contains the hexadecimal base address of the card. This is the lowest port address of the contiguously mapped RTI-1225 cards. The library routines use a 10-bit address. The RTI-1225 card should be mapped for an 8-bit I/O address. In software, set the most significant bits of the 10-bit address to 01 (for example, the RTI-1225 address 20 is entered as 120). The least significant four bits of the address must be 0. The routines do not work if the card is memory mapped.

The **CL** register specifies the number of contiguously mapped cards specified by this command (1 - 22).

The **DH** register contains the input multiplexer configuration:

01 = single-ended
02 = differential
03 = pseudo-differential

The **DL** register contains the A/D input range and output code:

00 = 0V to +10V, straight binary
01 = -5V to +5V, two's complement
02 = -5V to +5V, offset binary
03 = -10V to +10V, two's complement
04 = -10V to +10V, offset binary

The **AH** register contains the D/A channel 0 output range and input code:

00 = 0V to +10V, straight binary
01 = -5V to +5V, two's complement
02 = -5V to +5V, offset binary
03 = -10V to +10V, two's complement
04 = -10V to +10V, offset binary

The **AL** register contains the D/A channel 1 output range and input code:

00 = 0V to +10V, straight binary
01 = -5V to +5V, two's complement

02 = -5V to +5V, offset binary
03 = -10V to +10V, two's complement
04 = -10V to +10V, offset binary

Returned Value:

The AX register contains the returned integer error code:

0 = no error
-1 = board number out of range
-2 = invalid card count
    (CL>22 or the number of cards causes the base address of the last card to be less than that of the first card (wrap-around))
-3 = invalid multiplexer configuration
-4 = invalid A/D input range/output code
-5 = invalid D/A channel 0 input code/output range
-6 = invalid D/A channel 1 input code/output range
-7 = timeout error during reset

Example:

The following Assembly language program segment could be written to initialize the library for a group of four RTI-1225 cards after a group of three differently configured RTI-1225 cards have already been initialized. This group of cards is mapped contiguously beginning at address 120H. The cards are configured for differential inputs, 0 to +10V input range with straight binary code, and both output channels for 0 to +10V output range with straight binary code.

```
MOV   CH,3
MOV   BX,120H
MOV   CL,4
MOV   DH,02
MOV   DL,00
MOV   AH,00
MOV   AL,00
CALL  AD1225_INIT
```

Structure:	EXTRN AD1225_RESET:FAR **CALL AD1225_RESET**
Description:	This routine software resets all the RTI-1225 cards in the system. The multiplexer is set toselect input chan-nel 0, a wait for any pending conver-sions to complete or a timeout is performed. The D/A input registers are set to output a voltage of 0V. Note that the D/A output voltage may differ between hardware and software resets, depending on the user-selected hard-ware configuration.
Input Parameters:	None
Returned Value:	The AX register contains the returned integer error code: 0 = no error -7 = timeout error during reset
Example:	The following Assembly language program segment resets all RTI-1225 cards in a system.

```
CALL   AD1225_RESET
```

AD1225_TEST

Structure:

```
EXTRN     AD1225_TEST:FAR
MOV       DX,CARDNO
CALL      AD1225_TEST
```

Description:

This routine self-tests an RTI-1225 card. A voltage is read from the A/D converter and self-test is passed if there is no timeout error.

Input Parameters:

The **DX** register specifies the card to be tested (0 - 21).

Returned Value:

The **AX** register contains the returned integer error code:

0 = no error
-1 = invalid card selected
-7 = timeout error

Example:

The following Assembly language program segment tests the second RTI-1225 card in a system.

```
MOV   DX,1
CALL  AD1225_RESET
```

Analog Devices
RTI-1226
Analog Input Card
Assembly library

The Analog Devices RTI-1226 Analog Input Card is a general-purpose, low-cost analog I/O card. The RTI-1226 card uses an analog multiplexer to select among analog input channels that are connected to a 10-bit A/D converter. The input channels can be configured to be 16 single-ended or 8 differential inputs.

The STD-LIB RTI-1226 software modules are as follows:

AD1226_AIN:	Reads an input voltage and converts the A/D converter binary value to a reading rounded to the nearest millivolt (based on the default hardware configuration or the configuration set by AD1226_INIT).
AD1226_INIT:	Initializes the library software with the user-selected hardware configuration of the card. If this routine has been invoked properly, a call is made to the AD1226_RESET routine.
AD1226_RESET:	Performs a software reset of the card. The multiplexer is set to select input channel 0 and wait for any pending conversions to complete or a conversion busy bit timeout.
AD1226_TEST:	Performs a self-test on one card. The routine reads a voltage from the card and passes self-test if there is no timeout error.

NOTE: The RTI-1226 as-shipped mapping is at hexadecimal base address FFF0 in the memory map. To be compatible with the library, the card needs to be port mapped with an 8-bit address. The default 8-bit base address for the library is hexadecimal B0 which results in the card using ports BB through BD.

The Analog Devices convention is to define the base address of a card as an address with the last four bits being zero. The library has followed this convention. This means that an RTI-1226 card that has been remapped to the library default would be defined as a base address of 1B0 in the initialize command, but the physical ports of the card are actually located at 1BB through 1BD. When referring to contiguous mapping for RTI-1226 cards, the cards are mapped as though they occupied 16 ports per card, even though they only physically occupy three ports per card.

To change an RTI-1226 card from its as-shipped mapping to the library default mapping, make the following changes to the jumpers:

remove 59-60, 62-63, 51-52
install 61-63, 58-59, 53-55, 15-16

For technical questions concerning STD-LIB software for the Analog Devices RTI-1226, contact Pro-Log customer service. For technical questions concerning the operation of the Analog Devices RTI-1226 hardware, contact the appropriate Analog Devices customer service as follows:

STATE	PHONE NUMBER
Alaska	
Arizona	(714)-641-9391
California (southern)	Direct Sales Office
Colorado	
California (northern)	(408)-947-0633
Utah	Direct Sales Office
Delaware	(215)-643-7790
New Jersey	Direct Sales Office
Pennsylvania	
Maine	
Massachusetts	(617)-329-4700
New Hampshire	Factory
Rhode Island	
Illinois	
Indiana	
Iowa	
Kansas	
Kentucky	(312)-653-5000
Missouri	Direct Sales Office
North Dakota	
South Dakota	
Wisconsin	
All Other States	(617)-329-4700
	Factory

AD1226_AIN

Structure:

```
EXTRN  AD1226_AIN:FAR
MOV    DX,CARDNO
MOV    CX,CHANNEL
CALL   AD1226_AIN
```

Description:

This routine reads an input voltage from the selected channel. The binary reading of the A/D converter is translated to a binary reading in millivolts based on the user-selected input configuration.

Input Parameters:

The **DX** register specifies the card from which the reading is to be taken (0 - 21).

The **CX** register specifies the input channel to be read (0 - 15).

Returned Value:

The **AX** register contains the analog voltage read in millivolts (-10000 to + 9990) or contains an error code:

-32768 = CHANNEL number out of range
-32767 = board number out of range
-32766 = timeout error during conversion

Example:

The following Assembly language program segment could be written to read an analog voltage from input channel 2 of the fourth card in the system.

```
MOV    DX,3
MOV    CX,2
CALL   AD1226_AIN
```

Structure:

```
EXTRN AD1226_INIT:FAR
MOV     CH,BASENO
MOV     BX,BASEADR
MOV     CL,NOCARDS
MOV     DH,INCHAN
MOV     DL,ADCONF
CALL    AD1226_INIT
```

Description:

This routine initializes the RTI-1226 library routines with the user-selected hardware configuration of the card if different from the default condition. The card default initialization is for one RTI-1226 card in a system with a base address of 1B0, configured for 16 single-ended inputs of ±10V two's complement. There may be up to 22 of these cards in a system. When using more than one card, all cards with the same configuration are mapped contiguously and are initialized with the same routine call. If there are no parameter errors, the card is software reset with a call to the AD1226_RESET routine.

Input Parameters:

The **CH** register specifies the base number of the group of contiguous cards configured by this command (0 - 21).

The **BX** register contains the hexadecimal base address of the card. This is the lowest port address of the contiguously mapped RTI-1226 cards. The library routines use a 10-bit address. The RTI-1226 card should be mapped for an 8-bit I/O address. In software, set the most significant bits of the 10-bit address to 01 (for example, the RTI-1226 address 20 is entered as 120). The least significant four bits of the address must be 0. The routines do not work if the card is memory mapped.

The **CL** register specifies the number of contiguously mapped cards specified by this command (1 - 22).

The **DH** register contains the input multiplexer configuration:

01 = single-ended
02 = differential
03 = pseudo-differential

The **DL** register contains the A/D input range and output code:

00 = 0V to +10V, straight binary
01 = -5V to +5V, two's complement
02 = -5V to +5V, offset binary
03 = -10V to +10V, two's complement
04 = -10V to +10V, offset binary

Returned Value:

The **AX** register contains the returned integer error code:

0 = no error
-1 = board number out of range
-2 = invalid card count
(CL >22 or the number of cards causes the base address of the last card to be less than that of the first card (wrap around))
-3 = invalid multiplexer configuration
-4 = invalid A/D input range/output code
-7 = timeout error during reset

Example:

The following Assembly language segment could be written to initialize the library for a group of four RTI-1226 cards after a group of three differently configured RTI-1226 cards have already been initialized. This group of cards is mapped contiguously beginning at 8-bit

I/O address 0120H. The cards are configured for differential inputs, 0 to +10V input range with straight binary code.

```
MOV     CH,3
MOV     BX,0120H
MOV     CL,4
MOV     DH,2
MOV     DL,0
CALL    AD1226_INIT
```

Structure:

```
EXTRN  AD1226_RESET:FAR
CALL   AD1226_RESET
```

Description: This routine software resets all the RTI-1226 cards in the system. The multiplexer is set to select input channel 0, a wait for any pending conversions to complete or a timeout is performed.

Input Parameters: None

Returned Value: The AX register contains the returned integer error code:

0 = no error
-7 = timeout error during reset

Example: The following Assembly language segment resets all RTI-1226 cards in a system.

```
CALL   AD1226_RESET
```

Structure:

```
EXTRN  AD1226_TEST:FAR
MOV    DX,CARDNO
CALL   AD1226_TEST
```

Description:

This routine self-tests an RTI-1226 card. A voltage is read from the A/D converter and self-test is passed if there is no timeout error.

Input Parameters:

The **DX** register specifies the card to be tested (0 - 21).

Returned Value:

The **AX** register contains the returned integer error code:

 0 = no error
-1 = invalid card selected
-7 = timeout error

Example:

The following Assembly language program segment tests the second RTI-1226 card in a system.

```
MOV    DX,1
CALL   AD1226_TEST
```

Analog Devices
RTI-1260
Analog Input Card
Assembly library

The Analog Devices RTI-1260 Analog Input Card is a general-purpose, low-cost analog I/O card. The RTI-1260 card uses an analog multiplexer to select among analog input channels that are connected to a 12-bit A/D converter. The input channels can be configured to be 32 single-ended or 16 differential inputs.

The STD-LIB RTI-1260 software modules are as follows:

AD1260_AIN:	Reads an input voltage and converts the A/D converter binary value to a reading rounded to the nearest microvolt (based on the default hardware configuration or the configuration set by AD1260_INIT).
AD1260_INIT:	Initializes the library software with the user-selected hardware configuration of the card.
AD1260_RESET:	Performs a software reset of the card. The multiplexer is set to select input channel 0 and wait for any pending conversions to complete or a conversion busy bit timeout.
AD1260_TEST:	Performs a self-test on one card. The routine reads a voltage from the card and passes self-test if there is no timeout error.

NOTE: The RTI-1260 as-shipped mapping is at hexadecimal base address FFF0 in the memory map. To be compatible with the library, the card needs to be port mapped with an 8-bit address. The default 8-bit base address for the library is hexadecimal 30 which results in the card using ports 3B through 3D.

The Analog Devices convention is to define the base address of a card as an address with the last four bits being zero. The library has followed this convention. This means that an RTI-1260 card that has been remapped to the library default would be defined as a base address of 130 in the initialize command, but the physical ports of the card are actually located at 13B through 13D. When referring to contiguous mapping for RTI-1260 cards, the cards are mapped as though they occupied 16 ports per card, even though they only physically occupy three ports per card.

To change an RTI-1260 card from its as-shipped mapping to the library default mapping, make the following changes to the jumpers:

remove 48-50, 51-52, 44-45
install 49-50, 52-53, 45-47, 15-16, 7-8

For technical questions concerning STD-LIB software for the Analog Devices RTI-1260, contact Pro-Log customer service. For technical questions concerning the operation of the Analog Devices RTI-1260 hardware, contact the appropriate Analog Devices customer service as follows:

STATE	PHONE NUMBER
Alaska	
Arizona	(714)-641-9391
California (southern)	Direct Sales Office
Colorado	
California (northern)	(408)-947-0633
Utah	Direct Sales Office
Delaware	(215)-643-7790
New Jersey	Direct Sales Office
Pennsylvania	
Maine	
Massachusetts	(617)-329-4700
New Hampshire	Factory
Rhode Island	
Illinois	
Indiana	
Iowa	
Kansas	
Kentucky	(312)-653-5000
Missouri	Direct Sales Office
North Dakota	
South Dakota	
Wisconsin	
All Other States	(617)-329-4700
	Factory

AD1260_AIN

Structure:

```
EXTRN  1260_AIN:FAR
MOV    BH,CARDNO
MOV    BL,CHANNEL
CALL   AD1260_AIN
```

Description:

This routine reads an input voltage from the selected channel. The binary reading of the A/D converter is translated to a binary reading in microvolts based on the user-selected input configuration.

Input Parameters:

The **BH** register specifies the card from which the reading is to be taken (0 - 21).

The **BL** register specifies the input channel to be read (0 - 31).

Returned Value:

The **AX** register contains the analog voltage read in microvolts or contains an error code:

-20,000,000 = channel number out of range
-30,000,000 = board number out of range
-40,000,000 = timeout error during conversion

Example:

The following Assembly language program segment could be written to read an analog voltage from input channel 2 of the fourth card in the system.

```
MOV    BH,3
MOV    BL,2
CALL   AD1260_AIN
```

AD1260_INIT

Structure:

```
EXTRN AD1260_INIT:FAR
MOV     BH,BASENO
MOV     DX,BASEADR
MOV     BL,NOCARDS
MOV     AH,INCHAN
MOV     DI,GAINLOW
MOV     CX,GAINHIGH
MOV     AL,ADCONF
CALL    AD1260_INIT
```

Description:

This routine initializes the RTI-1260 library routines with the user-selected hardware configuration of the card if different from the default condition. The card default initialization is for one RTI-1260 card in a system with a base address of 130, gain equal one, configured for 32 single-ended inputs of ±10V offset binary. There may be up to 22 of these cards in a system. When using more than one card, all cards with the same configuration are mapped contiguously and are initialized with the same routine call.

Input Parameters:

The **BH** register specifies the base number of the group of contiguous cards configured by this command (0 - 21).

The **DX** register specifies the base address of the card. This is the lowest port address of the contiguously mapped RTI-1260 cards. The library routines use a 10-bit address. The RTI-1260 card should be mapped for an 8-bit I/O address. In software, set the most significant bits of the 10-bit address to 01 (for example, the RTI-1260 address 20 is entered as 120). The least significant four bits of the address must be 0. The routines do not work if the card is memory mapped.

0. The routines do not work if the card is memory mapped.

The **BL** register contains the number of contiguously mapped cards specified by this command (1 - 22).

The **AH** register contains the multiplexer configuration:

00 = single-ended
01 = differential
02 = pseudo-differential

The **CX:DI** registers contain the voltage gain on the A/D input multiplied by a factor of 10,000. The gain is multiplied by 10,000 to enable 12-bits of accuracy, hence a gain 1.2346 is entered as 12,346. The high order 16 bits is in CX and the low order 16 bits is contained in DI. The allowable gain range is 10000 - 10000000, this corresponds to a gain of 1V/V to 1000V/V.

The **AL** register contains the A/D input range and output code:

00 = 0V to +10V, straight binary
01 = -10V to +10V, offset binary
02 = -10V to +10V, two's complement

Returned Value:

The **AX** register contains the returned integer error code:

0 = no error
-1 = board number out of range
-2 = invalid card count
(nocards >22)
-3 = invalid multiplexer configuration
-4 = invalid A/D input range/output code
-5 = timeout error during reset
-6 = gain out of range

Example:

The following Assembly language program segment could be written to initialize the library for a group of four RTI-1260 cards after a group of three differently configured RTI-1260 cards have already been initialized. This group of cards is mapped contiguously beginning at 8-bit I/O address 0x20. The cards are configured for differential inputs, gain equal 1V/V, 0 to +10V input range with straight binary code.

```
MOV     BH,3
MOV     DX,0120H
MOV     BL,4
MOV     AH,1
MOV     DI,10000
MOV     CX,0
MOV     AL,0
CALL    AD1260_INIT
```

AD1260_RESET

Structure:

```
EXTRN  AD1260_RESET:FAR
CALL   AD1260_RESET
```

Description: This routine software resets board 0 and all initialized RTI-1260 cards in the system. The multiplexer is set to select input channel 0, a wait for any pending conversions to complete or a timeout is performed.

Input Parameters: None

Returned Value: The AX register contains the returned integer error code:

0 = no error
-5 = timeout error during reset

Example: The following Assembly language program segment resets all RTI-1260 cards in a system.

```
CALL AD1260_RESET
```

Structure:	EXTRN AD1260_TEST:FAR MOV BH,CARDNO; **CALL AD1260_TEST**
Description:	This routine self-tests an RTI-1260 card. A voltage is read from the A/D converter and self-test is passed if there is no timeout error.
Input Parameters:	The **BH** register specifies the card to be tested (0 - 21).
Returned Value:	The **AX** register contains the returned integer error code: 0 = no error -1 = invalid card selected -5 = timeout error
Example:	The following Assembly language program segment tests the second RTI-1260 card in a system.

```
MOV BH,1
CALL AD1260_TEST
```

Analog Devices
RTI-1262
Analog Output Card
Assembly library

The Analog Devices RTI-1262 Analog Output Card is a general-purpose, low-cost analog output card. There are four 12-bit D/A converters providing four output channels. Two of the four voltage outputs can be converted to a 4-20 milliamp current loop output by installing the correct hardware option.

The STD-LIB RTI-1262 software modules are as follows:

AD1262_AMPS:	Transforms the output current requested in the call from microamps to the nearest D/A code and writes an output current. The output current is in the range 4000 - 20000 microamps.
AD1262_AOUT:	Transforms the output voltage requested in the call from millivolts to the nearest D/A code and writes an output voltage.
AD1262_INIT:	Initializes the library software with the user-selected hardware configuration of the card.
AD1262_RESET:	Performs a software reset of the card. The multiplexer is set to select input channel 0 and wait for any pending conversions to complete or a conversion busy bit timeout. The D/A input registers are set to output a voltage of 0V.

NOTE: The RTI-1262 as-shipped mapping is at hexadecimal base address FFF0 in the memory map. To be compatible with the library, the card needs to be port mapped with an 8-bit address. The default 8-bit base address for the library is hexadecimal C0 which results in the card using ports C0 through C7.

The Analog Devices convention is to define the base address of a card as an address with the last four bits being zero. The library has followed this convention. This means that an RTI-1262 card that has been remapped to the library default would be defined as a base address of 1C0 in the initialize command, the physical ports of the card are actually located at 1C0 through 1C7. When referring to contiguous mapping for RTI-1262 cards, the cards are mapped as though they occupied 8 ports per card.

To change an RTI-1262 card from its as-shipped mapping to the library default mapping, make the following changes to the jumpers:

remove 58-60, 55-56, 51-52
install 59-60, 56-57, 52-53, 9-10, 1-2

For technical questions concerning STD-LIB software for the Analog Devices RTI-1262, contact Pro-Log customer service. For technical questions concerning the operation of the Analog Devices RTI-1262 hardware, contact the appropriate Analog Devices customer service as follows:

STATE	PHONE NUMBER
Alaska	
Arizona	(714)-641-9391
California (southern)	Direct Sales Office
Colorado	
California (northern)	(408)-947-0633
Utah	Direct Sales Office
Delaware	(215)-643-7790
New Jersey	Direct Sales Office
Pennsylvania	
Maine	
Massachusetts	(617)-329-4700
New Hampshire	Factory
Rhode Island	
Illinois	
Indiana	
Iowa	
Kansas	
Kentucky	(312)-653-5000
Missouri	Direct Sales Office
North Dakota	
South Dakota	
Wisconsin	
All Other States	(617)-329-4700
	Factory

AD1262_AMPS

Structure:

```
EXTRN  AD1262_AMPS:FAR
MOV    BH,CARDNO
MOV    BL,CHANNEL
MOV    CX,AMPS
CALL   AD1262_AMPS
```

Description:

This routine outputs a current to the selected channel. The selected current is input in micro-amps and converted to a binary program value for the V/I converter. The V/I converter must be installed for the channel and the channel must be configured for range 0 - 10 volts, straight binary encoding.

Input Parameters:

The **BH** register specifies the card from which the current is to be output (0 - 21).

The **BL** register specifies the output channel (0 - 3).

The **CX** register contains the current to be output in micro-amps (4000 - 20000).

Returned Value:

The **AX** register contains the error code:

 0 = no error
-1 = board number out of range
-2 = CHANNEL number out of range
-3 = current out of range

Example:

The following Assembly language program segment could be written to output a current of 5000 microamps to channel 0 of the fourth card in the system.

```
MOV    BH,3
MOV    BL,0
MOV      CX,5000
CALL AD1262_AMPS
```

AD1262_AOUT

Structure:

```
EXTRN  AD1262_AOUT:FAR
MOV    BH,CARDNO
MOV    BL,CHANNEL
MOV    AX,VOLTAGE
CALL   AD1262_AOUT
```

Description:

This routine outputs a voltage to the selected channel. The selected voltage is input in millivolts and converted to a binary program value for the D/A converter based on the user-selected output configuration for the channel.

Input Parameters:

The **BH** register specifies the card from which the voltage is to be output (0 - 21).

The **BL** register specifies the channel from which the voltage is to be output (0 - 3).

The **AX** register contains the analog voltage to be output in millivolts (-10000 to + 9990).

Returned Value:

The **AX** register contains the returned integer error code:

0 = no error
-1 = CHANNEL number out of range
-2 = board number out of range
-3 = voltage parameter out of range

Example:

The following Assembly language program segment could be written to output an analog voltage of 2V from output channel 1 of the fourth card in the system.

```
MOV    BH,3
MOV    BL,1
MOV    AX,2000
CALL   AD1262_AOUT
```

Structure:

```
EXTRN  AD1262_INIT:FAR
MOV    BH,BASENO
MOV    DX,BASEADR
MOV    BL,NOCARDS
MOV    AL,DA0CONF
MOV    AH,DA1CONF
MOV    CL,DA2CONF
MOV    CH,DA3CONF
CALL   AD1262_INIT
```

Description:

This routine initializes the RTI-1262 library routines with the user-selected hardware configuration of the card if different from the default condition. The card default initialization is for one RTI-1262 card in a system with a base address of 1C0, configured for -10V to +10V offset binary encoding.

Input Parameters:

The **BH** register contains the base number of the group of contiguous cards configured by this command (0 - 21).

The **DX** register contains a hexadecimal integer value specifying the base address of the card. This is the lowest port address of the contiguously mapped RTI-1262 cards. The library routines use a 10-bit address. The RTI-1262 card should be mapped for an 8-bit I/O address. In software, set the most significant bits of the 10-bit address to 01 (for example, the RTI-1262 address 20 is entered as 120). The least significant four bits of the address must be 0. The routines do not work if the card is memory mapped.

The **BL** register specifies the number of contiguously mapped cards specified by this command (1 - 22).

The AL register contains the D/A channel 0 output range and input code:

00 = 0V to +5V, straight binary
01 = 0V to +10V, straight binary
02 = -5V to +5V, offset binary
03 = -10V to +10V, offset binary

The AH register contains the D/A channel 1 output range and input code:

00 = 0V to +5V, straight binary
01 = 0V to +10V, straight binary
02 = -5V to +5V, offset binary
03 = -10V to +10V, offset binary

The CL register contains the D/A channel 2 output range and input code:

00 = 0V to +5V, straight binary
01 = 0V to +10V, straight binary
02 = -5V to +5V, offset binary
03 = -10V to +10V, offset binary

The CH register contains the D/A channel 3 output range and input code:

00 = 0V to +5V, straight binary
01 = 0V to +10V, straight binary
02 = -5V to +5V, offset binary
03 = -10V to +10V, offset binary

Returned Value:

The AX register contains the returned integer error code:

0 = no error
-1 = board number out of range
-2 = invalid card count
-3 = invalid D/A channel input code/output range

Example:

The following Assembly language program segment could be written to initialize the library for a group of four RTI-1262 cards after a group of three differently configured RTI-1262 cards have already been initialized. This group of cards is mapped contiguously beginning at 8-bit I/O address 20 hex. Output channels are configured for 0 to +10V output range with straight binary code.

```
MOV     BH,3
MOV     DX,0120H
MOV     BL,4
MOV     AL,1
MOV     AH,1
MOV     CL,1
MOV     CH,1
CALL    AD1262_INIT
```

Structure:

```
EXTRN AD1262_RESET:FAR
CALL AD1262_RESET
```

Description:

This routine software resets all the RTI-1262 cards in the system. The D/A input registers are set to output a voltage of 0V. Note that the D/A output voltage may differ between hardware and software resets, depending on the user-selected hardware configuration.

Input Parameters:

None

Returned Value:

The AX register contains the returned integer error code:

0 = no error

Example:

The following Assembly language program segment resets all RTI-1262 cards in a system.

```
CALL    AD1262_RESET
```

PRO-LOG

75XX,76XX,and 79XX Input/Output Cards

Assembly library

The Pro-Log input/output cards are general-purpose, low-cost analog I/O cards. The PL7000 library routines are general purpose I/O routines that can be used with the following Pro-Log cards: 7501, 7502, 7503, 7504, 7506, 7601A, 7602A, 7603A, 7604A, 7605, 7614, 7904, and 7914. Following the description of the STD-LIB routines is a description of each of the Pro-Log I/O cards that can be used with the library software. Shown below is a chart that shows which PL7000 routines can be used with each I/O card.

	7501	7502	7503	7504	7506	7601A	7602A	7603A	7604A	7605	7614	7904	7914
PL7000_INIT	*	*	*	*	*	*	*	*	*	*	*	*	*
PL7000_RDBIT			*		*	*		*	*	*	*	*	*
PL7000_RDPORT			o		o	*		*	*	*	*	*	*
PL7000_RESETPORT	*	*		*		*	*		*	*	*	*	*
PL7000_WRTBIT	*	*		*		*	*		*	*	*	*	*
PL7000_WRTBITRD									*	*	*	*	*
PL7000_WRTPORT	o	o		o		*	*		*	*	*	*	*
PL7000_WRTPORTRD									*	*	*	*	*

* = Routine can be used for card

o = Routine can be used for card, but other routine is typically preferred

The STD-LIB 7000 software modules are as follows:

PL7000_INIT:	Initializes the library software with the user-selected hardware configuration of the card.
PL7000_RDBIT:	Reads a bit from the selected port.
PL7000_RDPORT:	Reads an 8-bit value from the selected port.
PL7000_RESETPORT:	Software resets a group of contiguously mapped output ports by writing a 0 to all ports.
PL7000_WRTBIT:	Writes a 1 or a 0 to the selected bit of a port.
PL7000_WRTBITRD:	Writes a 1 or a 0 to the specified bit and port, then reads back the bit value to verify a proper write.
PL7000_WRTPORT:	Writes a specified 8-bit pattern to a port.
PL7000_WRTPORTRD:	Writes a specified 8-bit pattern to a port, then reads back an 8-bit value to verify a proper write.

Structure:

```
EXTRN PL7000_INIT:FAR
MOV     BH,BASENO
MOV     DX,BASEADR
MOV     CH,NOCARDS
MOV     CL,NOPORTS
CALL    PL7000_INIT
```

Description:

This routine initializes the PL7000 library for port I/O operations with the user selected hardware configuration if different from the default condition. The default is for one card (card 0) in a system containing two contiguously mapped ports with the lowest port address at hexadecimal 040. When using more than one card, all cards with the same configuration are mapped contiguously and are initialized with the same routine call.

Input Parameters:

The **BH** register contains an integer value specifying the base number of the group of contiguous cards configured by this command (0 -21).

The **DX** register contains a hexadecimal integer value specifying the base address of the card. This is the lowest port address of the contiguously mapped PL7000 cards.

The **CH** register specifies the number of contiguously mapped cards specified by this command (1 -22).

The **CL** register specifies the number of ports on each card (1 - 8).

Returned Value:	The AX register contains the returned integer error code:

0 = no error
-1 = board number out of range range
-2 = invalid card count (number of cards > 22, or the number of cards causes the base address of the last card to be less than that of the first card (wraparound)).
-3 = invalid number of ports

Example:

The following Assembly language program segment could be used to initialize the library for a group of four 7501 cards after a group of three differently configured cards have already been initialized. This group of cards is mapped contiguously beginning at 8-bit I/O address 20. The cards are configured for 2 ports per card.

```
MOV     BH,3
MOV     DX,0120H
MOV     CH,4
MOV     CL,2
CALL    PL7000_INIT
```

Structure:

```
EXTRN PL7000_RDBIT:FAR
MOV     BH,CARDNO
MOV     BL,PORTNO
MOV     CL,BITNO
CALL PL7000_RDBIT
```

Description:

This routine reads a bit from a port and returns 0, bit cleared, or 1 bit set.

Input Parameters:

The **BH** register specifies the card to read (0 - 21).

The **BL** register specifies the port to be read (0 - 7).

The **CL** register specifies the bit to be read (0 - 7).

Returned Value:

The **AX** register contains the returned bit value or integer error code:

 1 = bit is set
 0 = bit is cleared
-1 = card number out of range
-2 = port number out of range
-3 = bit number out of range

Example:

The following Assembly language program segment could be written to read bit four of the second port and the second card in the system.

```
MOV     BH,1
MOV     BL,1
MOV     CL,4
CALL    PL7000_RDBIT
```

PL7000_RDPORT

75XX/76XX/79XX Assembly library

Structure:

```
EXTRN  PL7000_RDPORT:FAR
MOV    BH,CARDNO
MOV    BL,PORTNO
CALL   PL7000_RDPORT
```

Description: This routine reads an 8-bit value from a port.

Input Parameters: The **BH** register specifies the card to be read from (0 - 21).

The **BL** register specifies the port to be read (0 - 7).

Returned Value: The **AX** register contains the port data (0 - 255) or the returned integer error code:

-1 = card number out of range
-2 = port number out of range

Example: The following Assembly language program segment could be written to read port 4 of the middle card in a three card system.

```
MOV    BH,1
MOV    BL,4
CALL   PL7000_RDPORT
```

PL7000_RESETPORT

Structure:

```
EXTRN  PL7000_RESETPORT
MOV    BH,BASENO
CALL   PL7000_RESETPORT
```

Description:

This routine software resets a group of contiguously mapped output ports. This group of cards was defined by PL7000_INIT. All ports in this group of contiguous cards should be output ports or input/output ports. A zero is written to all the ports.

Input Parameters:

The **BH** register specifies the base number of the group of contiguous cards reset by this command (0 - 21).

Returned Value:

The **AX** register contains the returned integer error code:

 0 = no error
-1 = base number out of range

Example:

The following Assembly language program segment resets a group of contiguously mapped cards beginning at card 0.

```
MOV    BH,BASENO
CALL   PL7000_RESETPORT
```

PL7000_WRTBIT

Structure:

```
EXTRN PL7000_WRTBIT:FAR
MOV     BH,CARDNO
MOV     BL,PORTNO
MOV     CH,BIT_DATA
MOV     CL,BITNO
CALL    PL7000_WRTBIT
```

Description:

This routine writes a 1 or a 0 to a specified bit of a port.

Input Parameters:

The **BH** register specifies the card to be written to (0 - 21).

The **BL** register specifies the port to be written to (0 - 7).

The **CH** register specifies the data to be written to the bit (0 or 1).

The **CL** register specifies the bit of the port being written to (0 - 7).

Returned Value:

The AX register contains the returned integer error code:

0 = no error
-1 = invalid card selected
-2 = port number out of range
-3 = bit number out of range
-4 = invalid write data

Example:

The following Assembly language program segment sets bit 0 of the second port on card 2 of the system.

```
MOV     BH,2
MOV     BL,1
MOV     CH,1
MOV     CL,0
CALL    PL7000_WRTBIT
```

PL7000_WRTBITRD

Structure:

```
EXTRN PL7000_WRTBITRD:FAR
MOV     BH,CARDNO
MOV     BL,PORTNO
MOV     CH,BIT_DATA
MOV     CL,BITNO
CALL    PL7000_WRTBITRD
```

Description:

This routine writes a 1 or a 0 to a specified bit of a port, then reads back the bit value to verify a proper write. This routine can only be used by cards with the readback capability.

Input Parameters:

The **BH** register specifies the card to be written to (0 - 21).

The **BL** register specifies the port to be written to (0 - 7).

The **CH** register specifies the data to be written to the bit (0 or 1).

The **CL** register specifies the bit of the port being written to (0 - 7).

Returned Value:

The AX register contains the returned integer error code:

0 = no error
-1 = invalid card selected
-2 = port number out of range
-3 = bit number out of range
-4 = invalid write data
-5 = read back error

Example:

The following Assembly language program segment sets bit 0 of the second port on card 2 of the system.

```
MOV     BH,2
MOV     BL,1
MOV     CH,1
MOV     CL,0
CALL PL7000_WRTBITRD
```

PL7000_WRTPORT

Structure:

```
EXTRN PL7000_WRTPORT:FAR
MOV     BH,CARDNO
MOV     BL,PORTNO
MOV     CH,PORTDATA
CALL    PL7000_WRTPORT
```

Description:

This routine writes an 8-bit pattern to a specified port.

Input Parameters:

The **BH** register specifies the card to be written to (0 - 21).

The **BL** register specifies the port to be written to (0 - 7).

The **CH** register specifies the data to be written to the port (0 hex - FF hex).

Returned Value:

The AX register contains the returned integer error code:

 0 = no error
-1 = invalid card selected
-2 = port number out of range

Example:

The following Assembly language program segment writes 4B hex to port 0 of the second card of the system.

```
MOV     BH,1
MOV     BL,0
MOV     CH,04BH
CALL PL7000_WRTPORT
```

PL7000_WRTPORTRD

Structure:

```
EXTRN PL7000_WRTPORTRD:FAR
MOV BH,CARDNO
MOV BL,PORTNO
MOV CH,PORTDATA
PL7000_WRTPORTRD
```

Description:

This routine writes an 8-bit pattern to a specified port, then reads back an 8-bit value to verify a proper write. This routine can only be used by cards with the readback capability.

Input Parameters:

The **BH** register specifies the card to be written to (0 - 21).

The **BL** register specifies the port to be written to (0 - 7).

The **CH** register specifies the data to be written to the port (0 hex - 0FF hex).

Returned Value:

The AX register contains the returned integer error code:

 0 = no error
-1 = invalid card selected
-2 = port number out of range
-3 = readback error

Example:

The following Assembly language program segment writes 4B hex to port 0 of the second card of the system.

```
MOV BH,1
MOV BL,0
MOV CH,04BH
CALL PL7000_WRTPORTRD
```

The following Pro-Log I/O cards may be used with the STD-LIB software:

7501 DC DRIVER CARD

The 7501 converts TTL level signals supplied by the STD BUS to latched, negative true, open-collector DC drive signals. The 7501 provides sixteen independent DC output circuits. Each open collector output circuit is capable of sinking up to 225 mA of current in the on state, and can withstand a 50V output level in the off state. The two output ports are cleared by SYSRESET* and are consecutively mapped at addresses 40 and 41 as shipped.

The 7501 uses the general-purpose I/O routines of STD-LIB as follows:

PL7000_INIT:	Initializes the library software with the user-selected hardware configuration.
PL7000_RESETPORT:	Software resets the output ports.
PL7000_WRTBIT:	Writes a 1 or a 0 to a specified bit of a port.

7502 SPST RELAY OUTPUT CARD

The 7502 consists of eight independent SPST dry reed relays controlled by a fully decoded, latched output port. The output port is cleared by SYSRESET* and is mapped at address 40 as shipped.

The 7502 uses the general-purpose I/O routines of STD-LIB as follows:

PL7000_INIT:	Initializes the library software with the user-selected hardware configuration.
PL7000_RESETPORT:	Software resets the output ports.
PL7000_WRTBIT:	Writes a 1 or a 0 to a specified bit of the port.

7503 OPTOISOLATED INPUT CARD

The 7503 converts an AC or DC input voltage into a TTL-level signal that is read by an on-card STD BUS input port. The 7503 provides eight independent AC/DC inputs. An optical coupling circuit isolates the circuits from each other and from the STD BUS system. The 7503 offers a choice of two input ranges. Range selection is independent for each input and is selected through the installation of wire jumpers. The low range responds to input voltages between 4.5 VRMS and 22 VRMS for 5V, 6V, and 12V applications. The high range responds to input voltages of between 20 VRMS and 80 VRMS and is useful for systems using 24V, 28V, and 48V. The input port is mapped at address 40 as shipped.

The 7503 uses the general-purpose I/O routines of STD-LIB as follows:

PL7000_INIT:	Initializes the library software with the user-selected hardware configuration.
PL7000_RDBIT:	Reads a bit from the port.

7504 TRIAC OUTPUT CARD

The 7504 consists of eight independent solid-state AC relays (triacs) controlled by a fully decoded, latched 8-bit output port. The 7504 allows the processor direct control of eight switched AC power circuits (40 - 280 VRMS and 47 - 63 Hz). The output port is cleared by SYSRESET*, and is mapped at address 40 as shipped.

The 7504 uses the general-purpose I/O routines of STD-LIB as follows:

PL7000_INIT:	Initializes the library software with the user-selected hardware configuration.
PL7000_RESETPORT:	Software resets the output ports.
PL7000_WRTBIT:	Writes a 1 or a 0 to a specified bit of the port.

7506 OPTOISOLATED INPUT CARD

The 7506 converts an AC or DC input voltage into a TTL-level signal that is read by an on-card STD BUS input port. The 7506 provides eight independent AC/DC inputs. An optical coupling circuit isolates the circuits from each other and from the STD BUS system. The 7506 offers a choice of two input ranges. Range selection is independent for each input and is selected through the installation of wire jumpers. The low range responds to input voltages between 70 VRMS and 150 VRMS for 120V applications. The high range responds to input voltages of between 140 VRMS and 280 VRMS and is useful for systems using 240V. The input port is mapped at address 40 as shipped.

The 7506 uses the general-purpose I/O routines of STD-LIB as follows:

PL7000_INIT:	Initializes the library software with the user-selected hardware configuration.
PL7000_RDBIT:	Reads a bit from the port.

7601A TTL INPUT/OUTPUT PORT CARD

The 7601A provides four 8-bit gated input ports (32 input lines) and four 8-bit latched output ports (32 output lines). The four consecutive independent input ports and output ports are mapped at address 00 - 03 as shipped. The output ports are cleared by SYSRESET*.

The 7601A uses the general-purpose I/O routines of STD-LIB as follows:

PL7000_INIT:	Initializes the library software with the user-selected hardware configuration.
PL7000_RDBIT:	Reads a bit from a port.
PL7000_RDPORT:	Reads an 8-bit value from a port.
PL7000_RESETPORT:	Software resets the output ports.
PL7000_WRTBIT:	Writes a 1 or a 0 to a specified bit of a port.
PL7000_WRTPORT:	Writes an 8-bit pattern to a port.

7602A TTL OUTPUT PORT CARD

The 7602A provides eight 8-bit latched output ports (64 lines). The consecutively mapped ports are cleared by SYSRESET*, and are mapped at address 00 -07 as shipped.

The 7602A uses the general-purpose I/O routines of STD-LIB as follows:

PL7000_INIT:	Initializes the library software with the user-selected hardware configuration.
PL7000_RESETPORT:	Software resets the output ports.
PL7000_WRTBIT:	Writes a 1 or a 0 to a specified bit of a port.
PL7000_WRTPORT:	Writes an 8-bit pattern to a port.

7603A TTL INPUT PORT CARD

The 7603A provides eight 8-bit gated input ports (64 input lines). The consecutive ports are mapped at address 00 - 07 as shipped.

The 7603A uses the general-purpose I/O routines of STD-LIB as follows:

PL7000_INIT:	Initializes the library software with the user-selected hardware configuration.
PL7000_RDBIT:	Reads a bit from a port.
PL7000_RDPORT:	Reads an 8-bit value from a port.

7604A TTL INPUT/OUTPUT CARD

The 7604A provides up to eight ports of TTL I/O. The eight ports can be individually configured as input, output, or output with readback. The consecutive ports are mapped at address 00 - 07 as shipped. The output ports are cleared by SYSRESET*.

The 7604A uses the general-purpose I/O routines of STD-LIB as follows:

PL7000_INIT:	Initializes the library software with the user-selected hardware configuration.
PL7000_RDBIT:	Reads a bit from a port.
PL7000_RDPORT:	Reads an 8-bit value from a port.
PL7000_RESETPORT:	Software resets the output ports.
PL7000_WRTBIT:	Writes a 1 or a 0 to a specified bit of a port.
PL7000_WRTBITRD:	Writes a 1 or a 0 to a specified bit of a port, and then reads back the value to verify a proper write.
PL7000_WRTPORT:	Writes an 8-bit pattern to a port.
PL7000_WRTPORTRD:	Writes an 8-bit pattern to a port, and then reads back the value to verify a proper write.

7605 PROGRAMMABLE TTL I/O CARD

The 7605 is a general-purpose TTL I/O card that provides four 8-bit I/O ports. Since each of the 32 I/O lines can be individually programmed as input, output, or output with readback, the 7605 is particularly useful when an external device requires more or less than eight lines or a mixture of inputs and outputs. The four consecutive ports are mapped at address 00 - 03 as shipped. The output ports are cleared by SYSRESET*.

The 7605 uses the general-purpose I/O routines of STD-LIB as follows:

PL7000_INIT:	Initializes the library software with the user-selected hardware configuration.
PL7000_RDBIT:	Reads a bit from a port.
PL7000_RDPORT:	Reads an 8-bit value from a port.
PL7000_RESETPORT:	Software resets the output ports.
PL7000_WRTBIT:	Writes a 1 or a 0 to a specified bit of a port.
PL7000_WRTBITRD:	Writes a 1 or a 0 to a specified bit of a port, and then reads back the value to verify a proper write.
PL7000_WRTPORT:	Writes an 8-bit pattern to a port.
PL7000_WRTPORTRD:	Writes an 8-bit pattern to a port, and then reads back the value to verify a proper write.

7614 64-LINE TTL I/O CARD

The 7614 provides eight 8-bit I/O ports, 64 I/O lines total. Each port can be configured as input, output, or output with readback. The 7614 operates in systems using an 8- or 10-bit I/O address. The I/O ports can be configured as four or eight ports, and are mapped at address 00 - 07 as shipped. The output ports are cleared by SYSRESET*.

The 7614 uses the general-purpose I/O routines of STD-LIB as follows:

PL7000_INIT:	Initializes the library software with the user-selected hardware configuration.
PL7000_RDBIT:	Reads a bit from a port.
PL7000_RDPORT:	Reads an 8-bit value from a port.
PL7000_RESETPORT:	Software resets the output ports.
PL7000_WRTBIT:	Writes a 1 or a 0 to a specified bit of a port.
PL7000_WRTBITRD:	Writes a 1 or a 0 to a specified bit of a port, and then reads back the value to verify a proper write.
PL7000_WRTPORT:	Writes an 8-bit pattern to a port.
PL7000_WRTPORTRD:	Writes an 8-bit pattern to a port, and then reads back the value to verify a proper write.

7904 DECODED I/O UTILITY CARD

The 7904 is a printed circuit card designed for prototyping I/O circuitry. It provides complete STD BUS buffering and decoding for I/O cards with input and output port strobe logic and access to memory page and CPU/memory control lines. The card is configured for eight consecutive I/O ports mapped at address 00 - 07 as shipped.

The 7904 can use the appropriate general-purpose I/O routines of STD-LIB as follows:

PL7000_INIT:	Initializes the library software with the user-selected hardware configuration.
PL7000_RDBIT:	Reads a bit from a port.
PL7000_RDPORT:	Reads an 8-bit value from a port.
PL7000_RESETPORT:	Software resets the output ports.
PL7000_WRTBIT:	Writes a 1 or a 0 to a specified bit of a port.
PL7000_WRTBITRD:	Writes a 1 or a 0 to a specified bit of a port, and then reads back the value to verify a proper write.
PL7000_WRTPORT:	Writes an 8-bit pattern to a port.
PL7000_WRTPORTRD:	Writes an 8-bit pattern to a port, and then reads back the value to verify a proper write.

7914 DECODED I/O UTILITY CARD

The 7914 is a printed circuit card designed for prototyping I/O circuitry. The card provides STD BUS buffering and I/O port select decoding. The card is configured for eight consecutive I/O ports mapped at address 0000 - 0007 as shipped.

The 7914 can use the appropriate general-purpose I/O routines of STD-LIB as follows:

PL7000_INIT:	Initializes the library software with the user-selected hardware configuration.
PL7000_RDBIT:	Reads a bit from a port.
PL7000_RDPORT:	Reads an 8-bit value from a port.
PL7000_RESETPORT:	Software resets the output ports.
PL7000_WRTBIT:	Writes a 1 or a 0 to a specified bit of a port.
PL7000_WRTBITRD:	Writes a 1 or a 0 to a specified bit of a port, and then reads back the value to verify a proper write.
PL7000_WRTPORT:	Writes an 8-bit pattern to a port.
PL7000_WRTPORTRD:	Writes an 8-bit pattern to a port, and then reads back the value to verify a proper write.

PRO-LOG
7303
Keyboard Display Card
Assembly library

The 7303 Keyboard Display Card is a general-purpose control panel card with data input and display capability. The card includes an 8-character alphanumeric display, keyboard with 24 program-definable keys plus system reset, an 8-bit binary LED display, and two rocker switches. The alphanumeric displays do not display lower case characters. The display routines convert characters to upper case.

The STD-LIB 7303 software modules are as follows:

PL7303_CHRSHFT:	Shifts the displayed characters to the left and brings in a new character at the right.
PL7303_CHRWR:	Writes an ASCII character to a specified display position.
PL7303_CURSOR:	Sets or removes one or more cursors from the display.
PL7303_GETKEY:	Waits for and then reads a keystroke. The routine waits for a new keystroke and then waits for key debounce.
PL7303_GETROCKR:	Reads a valid rocker switch state. The routine waits for any switch bouncing to finish.
PL7303_INIT:	Initializes the library software with the user-selected hardware configuration of the card. This routine also performs self-test functions: a write to the displays and LEDs for visual check, and a keyboard check for a "stuck key."

PL7303_KEYFETCH:	To be called 15 ms after the PL7303KEYSTAT routine. It returns a key if bouncing has died and a new keystroke has been detected.
PL7303_KEYSTAT:	Checks to see if a new key is currently closed. The routine does not indicate whether the depressed key has stopped bouncing.
PL7303_MESSAGE:	Shifts a message into the display from the right and continues shifting the message across until the display is blank.
PL7303_RCKRSTAT:	Returns the rocker switch status: open, closed, or changed.
PL7303_RESET:	Blanks all displays.
PL7303_SETLEDS:	Sets the state of the LEDs.
PL7303_TEST:	Performs a self-test of the card, including a write to the displays and LEDs for visual check, and a keyboard check for a "stuck key."
PL7303_WAIT:	Waits 15 ms for debouncing a key in the keyboard or a rocker switch.
PL7303_WORD:	Displays an 8-character (or shorter) word from left to right and blank fills any remaining portion of the display.

Structure:	EXTRN PL7303_CHRSHFT:FAR MOV DH,CHR **CALL PL7303_CHRSHFT**
Description:	This routine shifts the displayed characters to the left and brings in a new character at the right.
Input Parameters:	The **DH** register contains the hexadecimal representation of the ASCII character to be shifted into the display at the right.
Returned Value:	The **AX** register contains the error code: 0 = no error -1 = no 7303 installed -2 = invalid character selected
Example:	The following assembly language program segment could be written to shift the display left and bring in an "A" to the rightmost display.

```
MOV   DH,41H
CALL  PL7303_CHRSHFT
```

PL7303_CHRWR

Structure:

```
EXTRN   PL7303_CHRWR:FAR
MOV     DH,CHR
MOV     DL,DISPNO
CALL    PL7303_CHRWR
```

Description: This routine writes an ASCII character to the specified display.

Input Parameters: The **DH** register contains the hexadecimal representation of the ASCII character to be written to a display.

The **DL** register specifies the display to be written (0 - 7). The rightmost display is defined as display 0.

Returned Value: The **AX** register contains the error code:

 0 = no error
-1 = no 7303 installed
-2 = invalid display selected
-3 = invalid character selected

Example: The following assembly language program segment could be written to write an "A" to the third display from the left.

```
MOV   DH,41H
MOV   DL,5
CALL  PL7303_CHRWR
```

Structure:

```
EXTRN     PL7303_CURSOR:FAR
MOV       DH,DISPLAY
MOV       DL,ONOFF
CALL      PL7303_CURSOR
```

Description:

This routine writes a cursor (all segments lit) to the specified displays. With a valid ASCII character loaded to a display position, the cursor can also be displayed in that position. When the cursor is removed, the ASCII character reappears.

Input Parameters:

The **DH** register contains a binary number indicating which displays are to be written to.

The **DL** register contains a binary number specifying whether the associated cursor position is to be turned on or off (1 = on, 0 = off).

Returned Value:

The **AX** register contains the error code:

0 = no error
-1 = no 7303 installed

Example:

The following assembly language program segment could be written to write a cursor to the third display from the left and to turn off the rightmost cursor.

```
MOV   DH,21H
MOV   DL,20H
CALL  PL7303_CURSOR
```

Structure:	EXTRN PL7303_GETKEY:FAR CALL PL7303_GETKEY
Description:	This routine waits for a keystroke and then waits for key debounce. The routine then returns with the key that was pressed.
Input Parameters:	None
Returned Value:	The AX register contains the key that was detected or an error code: 0H-17H = key detected -1 = no 7303 installed
Example:	The following assembly language program segment could be written to read a key from the keypad. CALL PL7303_GETKEY

PL7303_GETROCKR

Structure:

```
EXTRN    PL7303_GETROCKR:FAR
CALL     PL7303_GETROCKR
```

Description:

This routine reads a valid rocker switch state. The routine waits for any switch bouncing to end.

Input Parameters:

None

Returned Value:

The AX register contains the rocker switch status:

 0 = both switches open
 1 = SW1 closed, SW2 open
 2 = SW2 closed, SW1 open
 3 = both switches closed
-1 = no 7303 installed

Example:

The following assembly language program segment could be written to get the valid status of rocker switch 2.

```
CALL  PL7303_GETROCKR
```

Structure:

```
EXTRN   PL7303_INIT:FAR
MOV     BX,BASEADR
CALL    PL7303_INIT
```

Description:

This routine initializes the 7303 library routines with the configuration of the card if different from the as-shipped condition (BASEADR = D0H). The storage for the previous condition of the rocker switches is initialized to open. There may be only one 7303 card in a system. This routine also lights all LEDs and all the segments on the displays for a one-second visual check of proper operation. A "stuck key" test is performed, in which the keyboard is checked to see that no key is depressed. All LEDs and displays are then blanked.

Input Parameters:

The **BX** register specifies the hexadecimal base address of the card. This is the lowest port address of the 7303 card. The library routines use a 10-bit address, even though the card can only be set to an 8-bit address. Set the two most significant bits of the 10-bit address to 01 (for example, 7303 address D0 is entered as 1D0).

Returned Value:	The AX register contains the error code:

0 = no error
-1 = no 7303 installed
other = row of stuck key
    (binary coded)
    bit 0=row1:keys 0-3
    bit 1=row2:keys 4-7
    bit 2=row3:keys 8-B
    bit 3=row4:keys C-F
    bit 4=row5:keys 10-13
    bit 5=row6:keys 14-17

Example: The following assembly language program segment could be written to initialize a 7303 card mapped at C0H.

```
MOV   BX,1C0H
CALL  PL7303_INIT
```

Structure:

```
EXTRN      PL7303_KEYFETCH:FAR
CALL       PL7303_KEYFETCH
```

Description: This routine reads the status of the keypad. It is called 15 ms after the PL7303_KEYSTAT routine.

Input Parameters: None

Returned Value: The AX register contains the returned keypad status. 0H - 17H for the key number if a valid key depression was detected.

-1 = no 7303 installed
-2 = no valid key detected

Example: The following assembly language program segment could be written to read the key status (typically written after PL7303_KEYSTAT and PL7303_WAIT routines or their equivalents have been called).

```
CALL  PL7303_KEYFETCH
```

PL7303_KEYSTAT

Structure:

```
EXTRN     PL7303_KEYSTAT:FAR
CALL      PL7303_KEYSTAT
```

Description:

This routine checks to see if a new key on the keypad is currently closed. The routine does not indicate whether the depressed key has stopped bouncing.

Input Parameters:

None

Returned Value:

The AX register contains the returned keypad status:

0 = no new key closed
1 = new key closed
-1 = no 7303 installed

Example:

The following assembly language program segment could be written to get the status of the keypad.

```
CALL  PL7303_KEYSTAT
```

PL7303_MESSAGE

Structure:

```
EXTRN       PL7303_MESSAGE:FAR

DATA        SEGMENT BYTE PUBLIC
ARRAY       DB SIZE DUP (?)
DB          00H
DATA        ENDS

MOV         AX,SEG DATA
MOV         ES,AX
LEA         DX,ARRAY
CALL        PL7303_MESSAGE
```

Description: This routine writes an ASCII message to the displays from the right. The message is shifted through the displays until all the displays are left blank. Any non-ASCII characters in the string are replaced with a space. A null character (00H) must be used as a string terminator.

Input Parameters: The **ES:DX** register pair contains a pointer to the message to be displayed. The message must have between 1 and 65535 characters.

Returned Value: The AX register contains the error code:

 0 = no error
-1 = no 7303 installed

Example: The following assembly language program segment could be written to scroll a 20-character message (stored at location ARRAY in SEG DATA) through the displays.

PL7303_MESSAGE

```
DATA      SEGMENT BYTE PUBLIC
ARRAY     DB 20 DUP (?)
DB        00H
DATA      ENDS

MOV       AX,SEG DATA
MOV       ES,AX
LEA       DX,ARRAY
CALL      PL7303_MESSAGE
```

Structure:

```
EXTRN   PL7303_RCKRSTAT:FAR
CALL    PL7303_RCKRSTAT
```

Description: This routine reads the current status of a rocker switch (open, closed, or changed). The routine does not indicate whether a rocker that has changed status since the last call has stopped bouncing.

Input Parameters: None

Returned Value: The AX register contains the rocker switch status:

1 = SW1 status change
2 = SW2 status change
3 = SW1 and SW2 status change
4 = no status change, both switches open
5 = no status change, SW1 closed, SW2 open
6 = no status change, SW2 closed, SW1 open
7 = no status change, both switches closed
-1 = no 7303 installed

Example: The following assembly language program segment could be written to get the status of the rocker switches.

```
CALL  PL7303_RCKRSTAT
```

PL7303_RESET

Structure:	EXTRN PL7303_RESET:FAR **CALL PL7303_RESET**
Description:	This routine software resets the 7303. All alphanumeric and LED displays are blanked.
Input Parameters:	None
Returned Value:	The AX register contains the error code: 0 = no error -1 = no 7303 installed
Example:	The following assembly language program segment resets the 7303. CALL PL7303_RESET

PL7303_SETLEDS

Structure:

```
EXTRN   PL7303_SETLEDS:FAR
MOV     DH,LEDS
MOV     DL,ONOFF
CALL    PL7303_SETLEDS
```

Description:

This routine sets the state of the LED displays. Note that many of the other 7303 routines can affect the state of the LEDs. The LEDs are restored to their previous condition after any of these routines are called.

Input Parameters:

The **DH** register contains a binary integer number indicating which LEDs are to be changed.

The **DL** register contains a binary integer number specifying the values for the LEDs that are being written (1 = on, 0 = off).

Returned Value:

The **AX** register contains the error code:

 0 = no error
-1 = no 7303 installed

Example:

The following assembly language program segment could be written to set the third LED from the left and clear the fifth LED from the left.

```
MOV  DH,28H
MOV  DL,20H
CALL PL7303_SETLEDS
```

PL7303_TEST

Structure:

```
EXTRN     PL7303_TEST:FAR
CALL      PL7303_TEST
```

Description:

This routine performs a self-test of the 7303 card. It lights all LEDs and all the segments on the displays for a one-second visual check of proper operation. A "stuck key" test is performed, in which the keyboard is checked to see that no key is depressed. All LEDs and displays are then blanked.

Input Parameters:

None

Returned Value:

The AX register contains the error code:

 0 = no error
-1 = no 7303 installed
other = row of stuck key
    (binary coded)
    bit 0=row1:keys 0-3
    bit 1=row2:keys 4-7
    bit 2=row3:keys 8-B
    bit 3=row4:keys C-F
    bit 4=row5:keys 10-13
    bit 5=row6:keys 14-17

Example:

The following assembly language program segment could be written to self-test a 7303 card.

```
CALL  PL7303_TEST
```

Structure:	EXTRN PL7303_WAIT:FAR **CALL PL7303_WAIT**
Description:	This routine waits 15 ms, the time required to debounce a keyboard switch or rocker switch.
Input Parameters:	None
Returned Value:	The AX register contains the error code: 0 = no error -1 = no 7303 installed
Example:	The following assembly language program segment waits 15 ms for a 7303 switch to debounce. CALL PL7303_WAIT

Structure:

```
EXTRN  PL7303_WORD:FAR

DATA      SEGMENT BYTE PUBLIC
ARRAY     DB 8 DUP (?)
DB        00H
DATA      ENDS

MOV       AX,SEG DATA
MOV       ES,AX
LEA       DX,ARRAY
CALL      PL7303_WORD
```

Description: This routine writes an ASCII 8-character (or shorter) word to the display from left to right. Blanks are written to any remaining portion of the display. A null character (00H) must be used as a string terminator if the word has less than 8 characters.

Input Parameters: The **ES:DX** register pair contains a pointer to the word to be displayed.

Returned Value: The **AX** register contains the error code:

0 = no error
-1 = no 7303 installed

Example: The following assembly language program segment could be written to write a word (stored at location ARRAY in SEG DATA) to the displays.

```
DATA      SEGMENT BYTE PUBLIC
ARRAY     DB 8 DUP (?)
DB        00H
```

```
DATA    ENDS

MOV     AX,SEG DATA
MOV     ES,AX
LEA     DX,ARRAY
CALL    PL7303_WORD
```

PRO-LOG
7312/7314A
Communications Card
Assembly library

The 7312 and 7314A Peripheral Interface Cards provide full-duplex RS-232-C serial channels for connection to terminals, printers, modems, and other peripherals or systems. Each card features two (7312) or four (7314A) serial channels, each with selectable DCE/DTE configuration using standard RS-232-C line drivers and receivers.

Up to 32 RS-232-C channels are supported by the STD-LIB software modules. Each channel can be separately configured for baud rate, data bits, stop bits, handshake, and parity. Input and output are interrupt driven with user-defined receive and transmit buffers.

The STD-LIB 7312 and 7314A software modules are as follows:

PL7314_BUFFLUSH:	Removes all characters from the transmit and/or receive buffer.
PL7314_CHANRST:	Initializes one RS-232-C channel.
PL7314_INIT:	Initializes the library software with the user-selected hardware configuration for all channels.
PL7314_INTEN:	Enables or disables receive, transmit, or break interrupts.
PL7314_RCV:	Reads one received character from the receive buffer.

PL7314_RCVCHK:	Checks to see if a received character is available in the receive buffer.
PL7314_SETBUF:	Allocates a transmit or receive buffer to be used in place of the default 8-character buffers allocated by PL7314_INIT.
PL7314_TEST:	Tests a serial channel by writing to and reading from the control registers for the channel.
PL7314_XMT:	Transmits a character by placing it in the transmit buffer.
PL7314_XMTBRK:	Transmits a break condition immediately.
PL7314_XMTCHK:	Checks to see if there is room in the transmit buffer for a character.
PL7314_XMTIMD:	Transmits one character immediately without using the transmit buffer.

NOTE: The 7312 and 7314A as-shipped mapping is at hexadecimal address 0B0 through 0B7. When one of these cards is installed in an STD-DOS system with the as-shipped mapping, STD-DOS initializes the card as communication ports COM1 through COM4. If the card is not used for STD-DOS communication ports, the library can simply re-initialize the card. To avoid the possibility of someone attempting to use a 7312 or 7314A card for two different purposes simultaneously, the default library address for a 7312 or 7314A card is different from the address used by STD-DOS. The library default addresss is hexadecimal 160 through 167.

To change a 7312-1 or 7314A-1 card from its as-shipped mapping to the library default mapping, make the following changes to the W1 jumper header:

remove 5-6
install 7-8
remove 9-10
install 13-14

PL7314_BUFFLUSH

Structure:

```
EXTRN   PL7314_BUFFLUSH:FAR
MOV     BH,CHAN
MOV     BL,XMT
MOV     CH,RCV
CALL    PL7314_BUFFLUSH
```

Description:

This routine removes all characters from the transmit and/or receive buffers.

Input Parameters:

The **BH** register specifies the number of the channel for which the transmit and/or receive buffer(s) is being flushed (0-31).

The **BL** register specifies whether the transmit buffer is to be flushed (BL=1) or not (BL=0).

The **CH** register specifies whether the receive buffer is to be flushed (CH=1) or not (CH=0).

Returned Value:

The **AX** register contains the returned integer error code:

 0 = no error
-1 = invalid channel selected

Example:

The following Assembly language program segment could be written to flush the transmit buffer for channel 4.

```
MOV   BH,4
MOV   BL,1
MOV   CH,0
CALL  PL7314_BUFFLUSH
```

Structure:

```
EXTRN  PL7314_CHANRST:FAR
MOV         BH,CHAN
MOV         SI,BAUD
MOV         CH,PARITY
MOV         DH,BITSCHAR
MOV         DL,STOPBIT
MOV         CL,HANDSHK
CALL        PL7314_CHANRST
```

Description:

This routine initializes a channel and enables the transmit, receive, and break interrupts for one channel of a 7312 or 7314A card. This routine must be called once for each serial channel in the system. The PL7314_INIT routine must be called before this routine can be called. The default 8-character transmit and receive buffers are allocated within the PL7314_INIT module. Different size buffers can be allocated by the PL7314_SETBUF module.

Input Parameters:

The **BH** register specifies the number of the channel being initialized (0-31).

The **SI** register specifies the baud rate (19200, 9600, 4800, 2400, 1200, 600, 300, 150, 110).

The **CH** register specifies parity (0 = no parity, 1 = odd parity, 2 = even parity).

The **DH** register specifies the number of data bits per character (5, 6, 7, 8).

The **DL** register specifies the number of stop bits (1 or 2).

The CL register specifies the handshake (0 = no handshake, 1 = CTS, 2 = XON/XOFF).

Returned Value:

The AX register contains the returned integer error code:

 0 = no error
-1 = error, channel cannot be
 initialized as requested

Example:

The following Assembly language program segment could be written to initialize serial channel 1 of a system. The channel is initialized to 9600 baud, no parity, 8 data bits per character, one stop bit, and XON/XOFF handshake protocol.

```
MOV     BH,1
MOV     SI,9600
MOV     CH,0
MOV     DH,8
MOV     BL,1
MOV     CL,2
CALL    PL7314_CHANRST
```

PL7314_INIT

Structure:

```
EXTRN     PL7314_INIT:FAR
BUFADR    DB MAXCHAN*48 DUP(?)
MOV       AH,MAXCHAN
MOV       AL,VECTLO
MOV       DX,BASEADR
LES       DI,BUFADR
CALL      PL7314_INIT
```

Description:

This routine initializes the 7312 or 7314A library routines with the user-selected hardware configuration for all serial channels. All 7312 or 7314A cards must be mapped contiguously. This routine checks that all indicated cards are properly installed and mapped. This routine must be called before any other 7314 module is called.

Input Parameters:

The **DX** register specifies the 10-bit base address of the cards. This is the lowest port address of the contiguously mapped 7312 and 7314 cards.

The **AH** register specifies the number of RS-232-C channels supported by 7312/7314A routines (1 - 32).

The **AL** register specifies the first interrupt vector available for receive and transmit interrupts. Starting with this interrupt, this software initializes and uses n consecutive interrupts, where n=the contents of the AH register divided by two (each pair of channels uses one interrupt).

ES:DI points to a buffer that can be used by the 7312/7314A software. The size of the buffer in bytes must be the contents of the AH register times 48 (48 bytes per channel are required).

Returned Value:	The AX register contains the returned integer error code:

0 = no error
n = number of requested channels
    that could not be initialized
    (n is a number from 1 to 32)

Example:

The following Assembly language program segment could be written to initialize the library for ten RS-232-C channels, located on two 7314A cards and one 7312 card. Channels 0-3 are located on a 7314A mapped at 100-107H, channels 4-7 on a 7314A mapped at 108H-10FH, and channels 8 and 9 on a 7312 mapped at 110-113H. Interrupt vectors C0-C4 are available for use by this module. Memory location BUFADR is a doubleword pointer to a 240-word RAM buffer.

```
BUFADR DW 240 DUP (?)
      .
      .

MOV         AH,10
MOV         AL,0C0H
MOV         DX,100H
LES         DI,BUFADR
CALL        PL7314_INIT
```

PL7314_INTEN

Structure:

```
EXTRN     PL7314_INTEN:FAR
MOV       CH,XMTINT
MOV       CL,RCVINT
MOV       BL,BRKINT
MOV       BH,CHAN
CALL      PL7314_INTEN
```

Description:

This routine enables or disables receive, transmit, and break interrupts for a channel..

Input Parameters:

The **CH** register specifies whether the transmitter interrupt for the channel is enabled (CH=1) or disabled (CH=0).

The **CL** register specifies whether the receiver interrupt for the channel is enabled (CL=1) or disabled (CL=0).

The **BL** register specifies whether the break condition interrupt for the channel is enabled (BL=1) or disabled (BL=0). When the break interrupt is enabled, the channel responds to a break interrupt by flushing the receive buffer and placing a CNTL-C character (03H) in the receive buffer.

The **BH** register specifies the number of the channel for which interrupt conditions are being defined (0-31).

Returned Value:

The **AX** register contains the returned integer error code:

0 = no error
-1 = invalid channel selected

Example:

The following Assembly language program segment could be written to enable transmitter and break condition interrupts and disable the receiver interrupt for channel 4.

```
MOV   CH,1
MOV   CL,0
MOV   BL,1
MOV   BH,4
CALL  PL7314_INTEN
```

PL7314_RCV

Structure:

```
EXTRN    PL7314_RCV:FAR
MOV      BH,CHAN
CALL     PL7314_RCV
```

Description:

This routine reads a received character from the buffer if one is present, otherwise it waits until a character is received. The routine is protected from never returning (if a character isn't received) by a timeout.

Input Parameters:

The **BH** register specifies the number of the channel from which the character is to be received (0-31).

Returned Value:

The **AX** register contains the returned status of the channel:

positive number = character received
-1 = invalid channel selected
-2 = timeout error
-3 = parity error
-4 = framing error
-5 = overrun error

Example:

The following Assembly language program segment could be written to wait for and then read a received character from channel 4.

```
MOV   BH,4
CALL  PL7314_RCV
```

Structure:	

```
EXTRN   PL7314_RCVCHK:FAR
MOV     BH,CHAN
CALL    PL7314_RCVCHK
```

Description:	This routine checks a serial channel to see if a character is available in the receive buffer.
Input Parameters:	The **BH** register specifies the number of the channel to be checked for a received character (0-31).
Returned Value:	The **AX** register contains the returned status of the channel: 0 = no character available positive number = number of characters available in the receive buffer -1 = invalid channel selected
Example:	The following Assembly language program segment could be written to check whether channel 4 has a received character available.

```
MOV   BH,4
CALL  PL7314_RCVCHK
```

PL7314_SETBUF

Structure:

```
EXTRN     PL7314_SETBUF:FAR

DATA      SEGMENT BYTE PUBLIC
ARRAY     DW SIZE/2 DUP (?)
DATA      ENDS

MOV       CX,SIZE
MOV       AX,SEG DATA
MOV       ES,AX
LEA       DX,ARRAY
MOV       BL,XMTRCV
MOV       BH,CHAN
CALL      PL7314_SETBUF
```

Description:

This routine allocates a transmit or receive buffer that is a different size than the 8-character default buffer allocated by PL7314_INIT. An n-character receive buffer requires 2 (n + 1) bytes and an n-character transmit buffer requires (n + 1) bytes.

Input Parameters:

The **CX** register specifies the size in bytes of the buffer to be allocated (1 - 32767).

The **ES:DX** register pair contains a pointer to the buffer area.

The **BL** register specifies whether the buffer is to be a transmit buffer or a receive buffer (0 = transmit, 1 = receive).

The **BH** register specifies the number of the channel for which the buffer is being defined (0-31).

Returned Value:	The **AX** register contains the returned integer error code:

```
 0 = no error
-1 = invalid channel selected
```

Example: The following Assembly language program segment could be written to allocate a 49-character receive buffer for channel 4.

```
DATA        SEGMENT BYTE PUBLIC
ARRAY       DW 50 DUP (?)
DATA        ENDS

MOV     CX,100
MOV     AX,SEG DATA
MOV     ES,AX
LEA     DX,ARRAY
MOV     BL,1
MOV     BH,4
CALL    PL7314_SETBUF
```

PL7314_TEST

Structure:

```
EXTRN     PL7314_TEST:FAR
MOV       BH,CHAN
CALL      PL7314_TEST
```

Description:

This routine performs a self-test on one serial channel. The routine tests the control registers of the serial chip to see if they can be written to and read from. The data registers aren't written to in this test to avoid changing any outputs of the channel.

Input Parameters:

The **BH** register specifies the number of the channel to be tested (0-31).

Returned Value:

The **AX** register contains the returned status of the channel:

 0 = no error
-1 = self test failed

Example:

The following Assembly language program segment could be written to test channel 4.

```
MOV   BH,4
CALL  PL7314_TEST
```

Structure:	See code below

```
EXTRN    PL7314_XMT:FAR
MOV      BH,CHAN
MOV      BL,CHR
CALL     PL7314_XMT
```

Description: This routine transmits a character by placing it in the transmit character buffer. If there is no room in the buffer, the routine waits until there is room or until a timeout indicates a system failure.

Input Parameters: The **BH** register specifies the number of the channel from which the character is to be transmitted (0-31).

The **BL** register contains the ASCII character to be transmitted.

Returned Value: The **AX** register contains the returned status of the channel:

 0 = no error
-1 = invalid channel selected
-2 = transmitter timeout error

Example: The following Assembly language program segment could be written to transmit an ASCII "A" from channel 4.

```
MOV   BH,4
MOV   BL,41H
CALL  PL7314_XMT
```

Structure:	

```
EXTRN    PL7314_XMTBRK:FAR
MOV      BH,CHAN
CALL     PL7314_XMTBRK
```

Description: This routine transmits a break condition immediately without waiting for the characters in the transmit buffer to be sent.

Input Parameters: The **BH** register specifies the number of the channel from which the break is to be transmitted (0-31).

Returned Value: The **AX** register contains the returned status of the channel:

0 = no error
-1 = invalid channel selected

Example: The following Assembly language program segment could be written to transmit an immediate break condition from channel 4.

```
MOV   BH,4
CALL  PL7314_XMTBRK
```

PL7314_XMTCHK

Structure:

```
EXTRN   PL7314_XMTCHK:FAR
MOV     BH,CHAN
CALL    PL7314_XMTCHK
```

Description: This routine checks to see if there is room in the transmit buffer for a character.

Input Parameters: The **BH** register specifies the number of the channel buffer to be checked (0-31).

Returned Value: The **AX** register contains the returned status of the channel:

0 = buffer has no room
positive number = number of empty
  locations in the buffer
-1 = invalid channel selected

Example: The following Assembly language program segment could be written to check whether channel 4 has room in the transmit buffer for another character.

```
MOV   BH,4
CALL  PL7314_XMTCHK
```

PL7314_XMTIMD

Structure:

```
EXTRN     PL7314_XMTIMD:FAR
MOV       BH,CHAN
MOV       BL,CHR
CALL      PL7314_XMTIMD
```

Description: This routine transmits a character immediately without placing it in the transmit character buffer.

Input Parameters: The **BH** register specifies the number of the channel from which the character is to be transmitted (0-31).

The **BL** register contains the ASCII character to be transmitted.

Returned Value: The **AX** register contains the returned status of the channel:

 0 = no error
-1 = invalid channel selected

Example: The following Assembly language program segment could be written to transmit an ASCII "A" from channel 4 without using the transmit buffer.

```
MOV   BH,4
MOV   BL,41H
CALL  PL7314_XMTIMD
```

PRO-LOG
7507/7508
TTL Input/Output Card
Assembly library

The 7507 and 7508 General Purpose Interface Cards allow direct connection to one (7507) or two (7508) 24-I/O module mounting racks available from Opto-22, Gordos, Crydom, etc. Module mounting racks are also available in 8-module and 16-module versions. However, the software assumes that a 7507 has only one rack connected to it, and that a 7508 has no more than two racks. This requirement is independent of rack size.

The STD-LIB 7507 and 7508 software modules are as follows:

PL7508_INIT:	Initializes the library software with the user-selected hardware configuration of the card.
PL7508_RACKPWR:	Turns the rack power supplied to the user on or off.
PL7508_READMOD:	Reads an input from a module or reads the status of an output to a module.
PL7508_RESET:	Turns the module rack power off and sets all outputs to the modules low.
PL7508_SETMOD:	Sets an output to a module to a 0 or a 1.
PL7508_TEST:	This is a routine that simply returns a pass indication for self-test, because the 7507 and 7508 cards can't be tested without changing the outputs.

PL7508_INIT

Structure:

```
EXTRN    PL7508_INIT:FAR
MOV      DX,BASEADR
MOV      BH,NORACK
CALL     PL7508_INIT
```

Description:

This routine initializes the 7507 or 7508 library routines with the user-selected hardware configuration of the card if different from the as-shipped condition (BASEADR = 50H, NORACK = 2). This routine also resets the card to all module rack power switched off, all outputs low (off), and initializes storage for the previous state of the output bits to low.

Input Parameters:

The **DX** register specifies the hexadecimal base address of the cards. This is the lowest port address of the contiguously mapped 7507 and 7508 cards. The library routines use a 10-bit address even if the card only recognizes 8 bits of address.

The **BH** register specifies the number of racks controlled by contiguously mapped cards (1 - 44).

Returned Value:

The **AX** register contains the returned integer error code:

0 = no error

Example:

The following assembly language program segment could be written to initialize the library for two 7508 cards, the first card mapped at 160-167 and the second mapped at 168-16F. These cards can control four 24-position racks.

```
MOV  DX,160H
MOV  BH,4
CALL PL7508_INIT
```

PL7508_RACKPWR

7507/7508 Assembly library

Structure:	`EXTRN PL7508_RACKPWR:FAR`
	`MOV BH,RACK`
	`MOV CH,ONOFF`
	`CALL PL7508_RACKPWR`

Description:	This routine turns rack power on or off.
Input Parameters:	The **BH** register specifies the rack power to be enabled or disabled (0 - 43). The **CH** register specifies whether rack power is to be turned on (CH = 1) or off (CH = 0).
Returned Value:	The AX register contains the error code: 0 = no error -1 = invalid rack selected
Example:	The following assembly language program segment could be written to turn on power to rack 7.

```
EXTRN    PL7508_RACKPWR:FAR
MOV      BH,RACK
MOV      CH,ONOFF
CALL     PL7508_RACKPWR
```

```
MOV   BH,7
MOV   CH,1
CALL  PL7508_RACKPWR
```

PL7508_READMOD

Structure:

```
EXTRN   PL7508_READMOD:FAR
MOV     BH,RACK
MOV     BL,MODULE
CALL    PL7508_READMOD
```

Description:

This routine reads an input module.

Input Parameters:

The **BH** register specifies the rack containing the module to be read (0 - 43).

The **BL** register specifies the input module to be read (0 - 23).

Returned Value:

The **AX** register contains the results of the read:

0 = bit clear
1 = bit set
-1 = invalid module or rack selected

Example:

The following assembly language program segment could be written to read input module 22 of the middle rack in a three rack system.

```
MOV  BH,1
MOV  BL,22
CALL PL7508_READMOD
```

Structure:	EXTRN PL7508_RESET:FAR **CALL PL7508_RESET**
Description:	This routine software resets all contiguously mapped 7507 or 7508 cards. All module outputs are set low (off), and all rack power is turned off.
Input Parameters:	None
Returned Value:	The AX register contains the returned integer error code: 0 = no error
Example:	The following assembly language program segment could be written to reset all contiguously mapped 7507 or 7508 cards. CALL PL7508_RESET

Structure:

```
EXTRN      PL7508_SETMOD:FAR
MOV        BH,RACK
MOV        BL,MODULE
MOV        CH,ONOFF
CALL       PL7508_SETMOD
```

Description:

This routine writes a 1 or a 0 to a module output.

Input Parameters:

The **BH** register specifies the rack containing the module to be written to (0 - 43).

The **BL** register specifies the output module to be written (0 - 23).

The **CH** register specifies the value to be written to the module (0 or 1).

Returned Value:

The **AX** register contains the error code:

0 = no error
-1 = invalid module or rack selected
-2 = readback error (always returned if rack power is off)

Example:

The following assembly language program segment could be written to set output 22 of the middle rack in a three-rack system to a 1 (turn on module 22 of rack 1).

```
MOV   BH,1
MOV   BL,22
MOV   CH,1
CALL  PL7508_SETMOD
```

Structure:	EXTRN **CALL**	PL7508_TEST:FAR **PL7508_TEST**

Description: This routine simply returns a pass indication for self-test, because the 7507 and 7508 cards can't be tested without changing the outputs. This routine is included because self-test routines are required for all cards in the library.

Input Parameters: None

Returned Value: The AX register contains the returned integer error code:

0 = no error

Example: The following assembly language program segment could be written to call a self-test routine for 7507 or 7508 cards.

```
CALL PL7508_TEST
```

PRO-LOG
7700
Memory Diagnostic Routines
Assembly library

The Memory Diagnostic routines are to designed to test all the memory cards in the STD-DOS system, this will give the user a high level of confidence that their system is fully functional. The memory diagnostic routines are:

PL7700_CHECKSUM:	Computes and reports the checksum of a specified area of the DOS system.
PL7700_MEMTEST:	Tests all system RAM and reports to the user the amount of memory (in kbytes) that was successfully tested.

PL7700_CHECKSUM

Structure:

```
EXTRN  PL7700_CHECKSUM:FAR
MOV    DX,SEGNO
MOV    CX,NUMBER_OF_PARGRAF
CALL   PL7700_CHECKSUM
```

Description:

This routine computes the two's complement checksum of a specified area of the DOS system. The checksum is reported as a long integer.

Input Parameters:

The **DX** register specifies the start segment of the area to be checksummed (offset = 0000).

The **CX** register specifies the number of 16-byte paragraphs to be added to the checksum.

Returned Value:

The **DX:AX** register pair contains the 32-bit checksum of the specified area of memory.

Example:

The following Assembly language program segment could be written to checksum the memory from C000:0000 to C000:FFFF.

```
MOV    DX,0C000H
MOV    CX,01000H
CALL   PL7700_CHECKSUM
```

Structure:	EXTRN PL7700_MEMTEST:FAR **CALL PL7700_MEMTEST**
Description:	This routine tests all contiguous RAM in the STD-DOS system. The test starts at segment 0000 and continues until the top of RAM or an error is detected. For each memory location, the memory test writes then verifies a 00 hex then an FF hex; once tested, each memory location is returned to its original value.
Input Parameters:	None
Returned Value:	The AX register contains the amount of RAM tested successfully (in kbytes).
Example:	The following Assembly language program segment could be written to test all contiguous RAM beginning at 0000:0000.

```
CALL     PL7700_MEMTEST
```

PRO-LOG
7863
CPU Card
Assembly library

The 7863 CPU card is equipped with a real-time clock device. The STD-LIB 7863 software provides the user with the ability to read or write to the clock, to wait for a specified period of time, and to set future time marks which later may be checked to see if the time has expired. All the 7863 library routines except SETALARM and SETMARK access the clock directly.

The STD-LIB 7863 software modules are as follows:

PL7863_CHKMARK:	Checks for expiration of a previously set time mark.
PL7863_GETDATE:	Gets the current date.
PL7863_GETTIME:	Gets the current time.
PL7863_GETWATCH:	Gets the date, time, and weekday in one call.
PL7863_INIT:	Sets the current date and time on the clock, then resets all time marks.
PL7863_SETALARM:	Sets an internal alarm mark to enable software to wait until a specific day and time.
PL7863_SETMARK:	Sets an internal time mark for a specified time interval from the current day and time.
PL7863_TEST:	Performs a self-test on the clock. The routine gets the time, waits a while, then gets the time again. It passes self-test if the time has increased.
PL7863_WAIT:	Waits for a specified period of time.
PL7863_WEEKDAY:	Gets the current day of the week.

The clock maintains the current year in the form 0-99. For convenience, the year as specified (and returned) by all routines is in the form 1980-2079. In addition, the clock maintains the current hour in 24-hour mode (0-23). This is reported to the caller in this fashion.

This clock supports time measurement with resolution of a hundredth of a second. For the PL7863_WAIT routine, time intervals shorter than the minimum measurement will be measured using software program loops.

The PL7863_SETALARM and PL7863_SETMARK (set interval) routines share the same internal structure. If the user attempts to set an interval mark using the same mark number as a previously set, unexpired alarm mark, the old mark will be overwritten by the new one. Marks may be reset at any time via calls to PL7863_SETMARK or PL7863_SETALARM. It is the user's responsibility not to overwrite an existing mark before it has expired.

Structure:	`EXTRN PL7863_CHKMARK:FAR`
	`MOV CX,MARKNUM`
	`CALL PL7863_CHKMARK`

Description: This routine checks for expiration of a previously set time mark. It verifies that the mark is set, then compares it with the current time. If the time is greater than the mark, the mark will be cleared.

Input Parameters: The CX register specifies the time mark to be checked (0-9).

Returned Value: The AX register contains the returned integer error code:

0 = interval has not expired
1 = interval has expired
-1 = invalid mark number
-2 = mark not set

Example: The following assembly language program segment could be written to check mark number 2 to see if it has expired.

```
MOV CX,2
CALL PL7863_CHKMARK
```

PL7863_GETDATE

Structure:

```
EXTRN     PL7863_GETDATE:FAR
DATA      SEGMENT BYTE PUBLIC
MONTH     DW ?
DAY       DW ?
DATA      ENDS

MOV       AX,SEG DATA
MOV       DS,AX
MOV       ES,AX
LEA       SI,MONTH
LEA       DI,DAY
CALL      PL7863_GETDATE
```

Description:

This routine gets the current date.

Input Parameters:

The **DS:SI** register pair contains a pointer to a word where the month will be stored.

The **ES:DI** register pair contains a pointer to a word where the day will be stored.

Returned Value:

The **AX** register contains the current year (1980-2079).

The word pointed by **DS:SI** receives the current month (1-12).

The word pointed by **ES:DI** receives the current day (1-31).

Example: The following assembly language program segment could be written to get the current date.

```
DATA        SEGMENT BYTE PUBLIC
MONTH       DW ?
DAY         DW ?
DATA        ENDS

MOV         AX,SEG DATA
MOV         DS,AX
MOV         ES,AX
LEA         SI,MONTH
LEA         DI,DAY
CALL        PL7863_GETDATE
```

PL7863_GETTIME

Structure:

```
EXTERN     PL7863_GETTIME:FAR
DATA       SEGMENT BYTE PUBLIC
MINUTES    DW ?
HUND       DW ?
DATA       ENDS

MOV        AX,SEG DATA
MOV        DS,AX
MOV        ES,AX
LEA        SI,MINUTES
LEA        DI,HUND
CALL       PL7863_GETTIME
```

Description: This routine gets the current time.

Input Parameters: The **DS:SI** register pair contains a pointer to a word where the minutes will be stored.

The **ES:DI** register pair contains a pointer to a word where the hundredths of a second will be stored.

Returned Value: The **AX** register contains the current hours (0-23).

The word pointed by **DS:SI** receives the current minutes (0-59).

The word pointed by **ES:DI** receives the current hundredths of a second (0-5999).

Example: The following assembly language program segment gets the current time.

```
DATA        SEGMENT BYTE PUBLIC
MINUTES     DW ?
HUND        DW ?
DATA        ENDS

MOV         AX,SEG DATA
MOV         DS,AX
MOV         ES,AX
LEA         SI,MINUTES
LEA         DI,HUND
CALL        PL7863_GETTIME
```

PL7863_GETWATCH

Structure:

```
EXTRN      PL7863_GETWATCH:FAR
DATA       SEGMENT BYTE PUBLIC
ARRAY      DW 8 DUP (?)
DATA       ENDS

MOV        AX,SEG DATA
MOV        ES,AX
LEA        DI,ARRAY
CALL       PL7863_GETWATCH
```

Description: This routine is used to obtain the date, time, and weekday in one call. Since the data cannot easily be passed, the caller should pass the address of an eight-word array.

Input Parameters: The **ES:DI** register pair contains a pointer to an eight-word array where the time data will be stored.

Returned Value: The **AX** register contains the returned integer error code:

0 = no error
-1 = pointer is illegal (pointer equals 0)

If AX = 0, the eight-word array pointed by ES:DI receives the following information in order:

year (1980-2079)
month (1-12)
day (1-31)
hours (0-23)
minutes (0-59)
seconds (0-59)
hundredths of seconds (0-99)
weekday (1 = Sunday, 2 = Monday, etc.)

Example:

The following assembly language program segment could be written to read all the time data.

```
DATA        SEGMENT BYTE PUBLIC
ARRAY       DW 8 DUP (?)
DATA        ENDS

MOV         AX,SEG DATA
MOV         ES,AX
LEA         DI,ARRAY
CALL        PL7863_GET WATCH
```

PL7863_INIT

Structure:

```
EXTRN    PL7863_INIT:FAR
MOV      DI,YEAR
MOV      CH,MONTH
MOV      CL,DAY
MOV      DH,HOURS
MOV      DL,MINUTES
MOV      SI,SECONDS
MOV      BL,HUNDREDTHS
CALL     PL7863_INIT
```

Description:

This routine sets the current date and time on the clock and clears any time marks that have been set. It also checks the clock with a call to the PL7863_TEST routine. Note that this routine will change the time and date known to the STD-DOS operating system.

Input Parameters:

The **DI** register specifies the current year (1980-2079).

The **CH** register specifies the current month (1-12).

The **CL** register specifies the current day (1-31).

The **DH** register specifies the current hours (0-23).

The **DL** register specifies the current minutes (0-59).

The **SI** register specifies the current seconds (0-59).

The **BL** register specifies the current hundredths of a second (0-99).

Returned Value: The AX register contains the returned integer error code:

 0 = no error
-1 = clock is not operational
-2 = invalid date
-3 = invalid time

Example: The following Assembly language program segment could be written to set the current time to 07:00:00, September 1, 1986. The 7863 library software is also initialized.

```
MOV         DI,1986
MOV         CH,9
MOV         CL,1
MOV         DH,7
MOV         DL,0
MOV         SI,0
MOV         BL,0
CALL        PL7863_INIT
```

Structure:

```
EXTRN   PL7863_SETALARM:FAR
MOV     CX,MARKNUM
MOV     BL,WEEKDAY
MOV     DH,HOURS
MOV     DL,MINUTES
MOV     SI,SECONDS
CALL    PL7863_SETALARM
```

Description:

This routine sets an alarm mark for the specified time. PL7863_CHKMARK can be used to determine if the alarm has occurred. Up to 10 time marks may be set at a time. The 10 time marks may be set by either this routine or the PL7863_SETMARK routine, so care must be taken not to overwrite marks set by PL7863_SETMARK. If the specified time is earlier than the current time, the alarm will be set for the next coming week.

Input Parameters:

The **CX** register specifies the time mark to be set (0-9).

The **BL** register specifies the day of the week (1-7) of the alarm time.

The **DH** register specifies the hour (0-23) of the alarm time.

The **DL** register specifies the minutes (0-59) of the alarm time.

The **SI** register specifies the seconds (0-59) of the alarm time.

Returned Value:

The AX register contains the returned integer error code:

 0 = no error
-1 = invalid mark number
-2 = invalid interval

Example:

The following assembly language program segment could be written to set mark number 2 for Monday, 7 a.m.

```
MOV         CX,2
MOV         BL,2
MOV         DH,7
MOV         DL,0
MOV         SI,0
CALL        PL7863_SETALARM
```

PL7863_SETMARK

Structure:

```
EXTRN     PL7863_SETMARK:FAR
MOV       CX,MARKNUM
MOV       BL,DAYS
MOV       DH,HOURS
MOV       DL,MINUTES
MOV       SI,SECONDS
CALL      PL7863_SETMARK
```

Description:

This routine sets a time mark, measured from the current time, whose expiration can be checked via calls to PL7863_CHKMARK. Up to 10 time marks may be set at a time. These ten time marks may be set by this routine or the PL7863_SETALARM routine. The maximum time for a mark interval is one week.

Input Parameters:

The CX register specifies the time mark to be set (0-9).

The BL register specifies the number of days (0-6) in the time interval.

The DH register specifies the number of hours (0-23) in the time interval.

The DL register the number of minutes (0-59) in the time interval.

The SI register specifies the number of seconds (0-59) in the time interval.

Returned Value:

The AX register contains the returned integer error code:

0 = no error
-1 = invalid mark number
-2 = invalid time interval

Example:

The following assembly language program segment could be written to set mark number 3 for one minute from now.

```
MOV     CX,3
MOV     BL,0
MOV     DH,0
MOV     DL,1
MOV     SI,0
CALL    PL7863_SETMARK
```

PL7863_TEST

Structure:

```
EXTRN        PL7863_TEST:FAR
CALL         PL7863_TEST
```

Description:

This routine determines if the clock is operational. It will get the time, wait a while (known to be more than a hundredth of a second), and then get the time again. The clock is presumed to be operating correctly if the time has increased.

Input Parameters:

None

Returned Value:

The AX register contains the returned integer error code:

 0 = no error
-1 = clock is not operational

Example:

The following assembly language program segment tests the clock.

```
CALL         PL7863_TEST
```

PL7863_WAIT

Structure:

```
EXTRN     PL7863_WAIT:FAR
MOV       SI,SECONDS
MOV       AX,MILLI
MOV       CX,MICRO
CALL      PL7863_WAIT
```

Description:

This routine can be used to wait for a specified period of time. The maximum waiting time interval is 60 seconds.

Input Parameters:

The **SI** register specifies the number of seconds (0-59) in the wait interval.

The **AX** register specifies the number of milliseconds (0-999) in the wait interval.

The **CX** register specifies the number of microseconds (0-999) in the wait interval.

Returned Value:

The AX register contains the returned integer error code:

 0 = no error
-1 = invalid interval

Example:

The following assembly language program segment could be written to wait for 2.5 seconds.

```
MOV       SI,2
MOV       AX,500
MOV       CX,0
CALL      PL7863_WAIT
```

PL7863_WEEKDAY

Structure:

```
EXTRN        PL7863_WEEKDAY:FAR
CALL         PL7863_WEEKDAY
```

Description: This routine gets the current day of the week.

Input Parameters: None

Returned Value: The AX register contains the current day of the week (1-7), 1 = Sunday, 2 = Monday, etc.

Example: The following assembly language program segment gets the current day of the week.

```
CALL         PL7863_WEEKDAY
```

PRO-LOG
7888
CPU Card
Assembly library

The 7888 CPU card is equipped with a real-time clock device. The STD-LIB 7888 software provides the user with the ability to read or write to the clock, to wait for a specified period of time, and to set future time marks which later may be checked to see if the time has expired. All the 7888 library routines except SETALARM and SETMARK access the clock directly.

The STD-LIB 7888 software modules are as follows:

PL7888_CHKMARK:	Checks for expiration of a previously set time mark.
PL7888_GETDATE:	Gets the current date.
PL7888_GETTIME:	Gets the current time.
PL7888_GETWATCH:	Gets the date, time, and weekday in one call.
PL7888_INIT:	Sets the current date and time on the clock, then resets all time marks.
PL7888_SETALARM:	Sets an internal alarm mark to enable software to wait until a specific day and time..
PL7888_SETMARK:	Sets an internal future time mark for a specified time interval from the current day and time.
PL7888_TEST:	Performs a self-test on the clock by reading the status bit.
PL7888_WAIT:	Waits for a specified period of time.
PL7888_WEEKDAY:	Gets the current day of the week.

The clock maintains the current year in the form 0-99. For convenience, the year as specified (and returned) by all routines is in the form 1980-2079. In addition, the clock maintains the current hour in 24-hour mode (0-23). This is reported to the caller in this fashion.

This clock supports time measurement with resolution of a hundredth of a second. For the PL7888_WAIT routine, time intervals shorter than the minimum measurement will be measured using software program loops.

The PL7888_SETALARM and PL7888_SETMARK (set interval) routines share the same internal structure. If the user attempts to set an interval mark using the same mark number as a previously set, unexpired alarm mark, the old mark will be overwritten by the new one. Marks may be reset at any time via calls to PL7888_SETMARK or PL7888_SETALARM. It is the user's responsibility not to overwrite an existing mark before it has expired.

The watch chip on the 7888 CPU board will only resolve times down to one second. However, a few subroutine linkages feature times of hundredths of a second. This has been done to make 7863 CPU board support and 7888 CPU board support as similar as possible.

PL7888_CHKMARK

Structure:

```
EXTRN       PL7888_CHKMARK:FAR
MOV         CX,MARKNUM
CALL        PL7888_CHKMARK
```

Description: This routine checks for expiration of a previously set time mark. It verifies that the mark is set, then compares it with the current time. If the time is greater than the mark, the mark will be cleared.

Input Parameters: The CX register specifies the time mark to be checked (0-9).

Returned Value: The AX register contains the returned integer error code:

0 = interval has not expired
1 = interval has expired
-1 = invalid mark number
-2 = mark not set

Example: The following assembly language program segment could be written to check mark number 2 to see if it has expired.

```
MOV CX,2
CALL PL7888_CHKMARK
```

PL7888_GETDATE

Structure:

```
EXTRN     PL7888_GETDATE:FAR
DATA      SEGMENT BYTE PUBLIC
MONTH     DW ?
DAY       DW ?
DATA      ENDS

MOV       AX,SEG DATA
MOV       DS,AX
MOV       ES,AX
LEA       SI,MONTH
LEA       DI,DAY
CALL      PL7888_GETDATE
```

Description: This routine gets the current date.

Input Parameters: The **DS:SI** register pair contains a pointer to a word where the month will be stored.

The **ES:DI** register pair contains a pointer to a word where the day will be stored.

Returned Value: The **AX** register contains the current year (1980-2079).

The word pointed by **DS:SI** receives the current month (1-12).

The word pointed by **ES:DI** receives the current day (1-31).

Example:

The following assembly language program segment could be written to get the current date.

```
DATA        SEGMENT BYTE PUBLIC
MONTH       DW ?
DAY         DW ?
DATA        ENDS

MOV         AX,SEG DATA
MOV         DS,AX
MOV         ES,AX
LEA         SI,MONTH
LEA         DI,DAY
CALL        PL7888_GETDATE
```

PL7888_GETTIME

Structure:

```
EXTERN     PL7888_GETTIME:FAR
DATA       SEGMENT BYTE PUBLIC
MINUTES    DW ?
HUND       DW ?
DATA       ENDS

MOV        AX,SEG DATA
MOV        DS,AX
MOV        ES,AX
LEA        SI,MINUTES
LEA        DI,HUND
CALL       PL7888_GETTIME
```

Description:

This routine gets the current time. Due to linkage limitations, the returned time will have a "hundredths of a second" field rather than a "seconds" field.

Input Parameters:

The **DS:SI** register pair contains a pointer to a word where the minutes will be stored.

The **ES:DI** register pair contains a pointer to a word where the hundredths of a second will be stored.

Returned Value:

The **AX** register contains the current hours (0-23).

The word pointed by **DS:SI** receives the current minutes (0-59).

The word pointed by **ES:DI** receives the current hundredths of a second (0-5999).

Example:

The following assembly language program segment gets the current time.

```
DATA        SEGMENT BYTE PUBLIC
MINUTES     DW ?
HUND        DW ?
DATA        ENDS

MOV         AX,SEG DATA
MOV         DS,AX
MOV         ES,AX
LEA         SI,MINUTES
LEA         DI,HUND
CALL        PL7888_GETTIME
```

PL7888_GETWATCH

Structure:

```
EXTRN     PL7888_GETWATCH:FAR
DATA      SEGMENT BYTE PUBLIC
ARRAY     DW 8 DUP (?)
DATA      ENDS

MOV       AX,SEG DATA
MOV       ES,AX
LEA       DI,ARRAY
CALL      PL7888_GETWATCH
```

Description: This routine is used to obtain the date, time, and weekday in one call. Since the data cannot easily be passed, the caller should pass the address of an eight-word array.

Input Parameters: The **ES:DI** register pair contains a pointer to an eight-word array where the time data will be stored.

Returned Value: The **AX** register contains the returned integer error code:

0 = no error
-1 = pointer is illegal (pointer equals 0)

If AX = 0, the eight-word array pointed by ES:DI receives the following information in order:

year (1980-2079)
month (1-12)
day (1-31)
hours (0-23)
minutes (0-59)
seconds (0-59)
hundredths of seconds (0-99)
weekday (1 = Sunday, 2 = Monday, etc.)

Example:

The following assembly language program segment could be written to read all the time data.

```
DATA      SEGMENT BYTE PUBLIC
ARRAY     DW 8 DUP (?)
DATA      ENDS

MOV       AX,SEG DATA
MOV       ES,AX
LEA       DI,ARRAY
CALL      PL7888_GET WATCH
```

PL7888_INIT

Structure:

```
EXTRN    PL7888_INIT:FAR
MOV      DI,YEAR
MOV      CHI,MONTH
MOV      CL,DAY
MOV      DH,HOURS
MOV      DL,MINUTES
MOV      SI,SECONDS
MOV      BL,HUNDREDTHS
CALL     PL7888_INIT
```

Description:

This routine sets the current date and time on the clock and clears any time marks that have been set. It also checks the clock with a call to the PL7888_TEST routine. Note that this routine will change the time and date known to the STD-DOS operating system when the system is subsequently reset.

Input Parameters:

The **DI** register specifies the current year (1980-2079).

The **CH** register specifies the current month (1-12).

The **CL** register specifies the current day (1-31).

The **DH** register specifies the current hours (0-23).

The **DL** register specifies the current minutes (0-59).

The **SI** register specifies the current seconds (0-59).

The **BL** register specifies the current hundredths of a second (0-99).

Returned Value:

The AX register contains the returned integer error code:

0 = no error
-1 = clock is not operational
-2 = invalid date
-3 = invalid time

Example:

The following Assembly language program segment could be written to set the current time to 07:00:00, September 1, 1986. The 7888 library software is also initialized.

```
MOV         DI,1986
MOV         CH,9
MOV         CL,1
MOV         DH,7
MOV         DL,0
MOV         SI,0
MOV         BL,0
CALL        PL7888_INIT
```

PL7888_SETALARM

Structure:

```
EXTRN     PL7888_SETALARM:FAR
MOV       CX,MARKNUM
MOV       BL,WEEKDAY
MOV       DH,HOURS
MOV       DL,MINUTES
MOV       SI,SECONDS
CALL      PL7888_SETALARM
```

Description:

This routine sets an alarm mark for the specified time. PL7888_CHKMARK can be used to determine if the alarm has occurred. Up to 10 time marks may be set at a time. The 10 time marks may be set by either this routine or the PL7888_SETMARK routine, so care must be taken not to overwrite marks set by PL7888_SETMARK. If the specified time is earlier than the current time, the alarm will be set for the next coming week.

Input Parameters:

The **CX** register specifies the time mark to be set (0-9).

The **BL** register specifies the day of the week (1-7) of the alarm time.

The **DH** register specifies the hour (0-23) of the alarm time.

The **DL** register specifies the minutes (0-59) of the alarm time.

The **SI** register specifies the seconds (0-59) of the alarm time.

Returned Value:

The AX register contains the returned integer error code:

 0 = no error
-1 = invalid mark number
-2 = invalid interval

Example:

The following assembly language program segment could be written to set mark number 2 for Monday, 7 a.m.

```
MOV         CX,2
MOV         BL,2
MOV         DH,7
MOV         DL,0
MOV         SI,0
CALL        PL7888_SETALARM
```

PL7888_SETMARK

Structure:

```
EXTRN     PL7888_SETMARK:FAR
MOV       CX,MARKNUM
MOV       BL,DAYS
MOV       DH,HOURS
MOV       DL,MINUTES
MOV       SI,SECONDS
CALL      PL7888_SETMARK
```

Description:

This routine sets a time mark, measured from the current time, whose expiration can be checked via calls to PL7888_CHKMARK. Up to 10 time marks may be set at a time. These ten time marks may be set by this routine or the PL7888_SETALARM routine. The maximum time for a mark interval is one week.

Input Parameters:

The **CX** register specifies the time mark to be set (0-9).

The **BL** register specifies the number of days (0-6) in the time interval.

The **DH** register specifies the number of hours (0-23) in the time interval.

The **DL** register the number of minutes (0-59) in the time interval.

The **SI** register specifies the number of seconds (0-59) in the time interval.

Returned Value:

The AX register contains the returned integer error code:

0 = no error
-1 = invalid mark number
-2 = invalid time interval

Example:

The following assembly language program segment could be written to set mark number 3 for one minute from now.

```
MOV         CX,3
MOV         BL,0
MOV         DH,0
MOV         DL,1
MOV         SI,0
CALL        PL7888_SETMARK
```

PL7888_TEST

Structure:	EXTRN PL7888_TEST:FAR CALL PL7888_TEST
Description:	This routine determines if the clock is operational by reading the clock status.
Input Parameters:	None
Returned Value:	The AX register contains the returned integer error code: 0 = no error -1 = clock is not operational
Example:	The following assembly language program segment tests the clock. CALL PL7888_TEST

PL7888_WAIT

Structure:

```
EXTRN       PL7888_WAIT:FAR
MOV         SI,SECONDS
MOV         AX,MILLI
MOV         CX,MICRO
CALL        PL7888_WAIT
```

Description:

This routine can be used to wait for a specified period of time. The maximum waiting time interval is 60 seconds.

Input Parameters:

The **SI** register specifies the number of seconds (0-59) in the wait interval.

The **AX** register specifies the number of milliseconds (0-999) in the wait interval.

The **CX** register specifies the number of microseconds (0-999) in the wait interval.

Returned Value:

The **AX** register contains the returned integer error code:

 0 = no error
-1 = invalid interval

Example:

The following assembly language program segment could be written to wait for 2.5 seconds.

```
MOV         SI,2
MOV         AX,500
MOV         CX,0
CALL        PL7888_WAIT
```

PL7888_WEEKDAY

Structure:

```
EXTRN     PL7888_WEEKDAY:FAR
CALL      PL7888_WEEKDAY
```

Description: This routine gets the current day of the week.

Input Parameters: None

Returned Value: The **AX** register contains the current day of the week (1-7), 1 = Sunday, 2 = Monday, etc.

Example: The following assembly language program segment gets the current day of the week.

```
CALL      PL7888_WEEKDAY
```

Pro-Log Corporation
2560 Garden Road, Monterey, CA 93940
Phone: (408) 372-4593, (800) 538-9570
Telex: 171879, TWX: 910-360-7082

112626-003 5/87

MP MASTER
MA34218